SCALING SOLUTIONS TO MARKOV DECISION PROBLEMS
BY PENG ZANG

A Thesis
Presented to
The Academic Faculty

by

Peng Zang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
December 2011

SCALING SOLUTIONS TO MARKOV DECISION PROBLEMS
BY PENG ZANG

Approved by:

Charles Isbell, Advisor
School of Interactive Computing
Georgia Institute of Technology

Andrea Thomaz
School of Interactive Computing
Georgia Institute of Technology

Alexander Gray
School of Computational Science & Engineering
Georgia Institute of Technology

Mike Stilman
School of Interactive Computing
Georgia Institute of Technology

Michael L. Littman
Department of Computer Science
Rutgers University

Date Approved: 28 October 2011

To my dad and mom, Zang Zhitong and Yang Xiuqing.

ACKNOWLEDGEMENTS

Research is often a long process of exploration filled with twists and turns. Mine has been no exception. This dissertation would not have been possible without the help and support of numerous others.

I would like to thank my adviser, Charles Isbell, first and foremost. His enduring support of my work and steady guidance throughout the years have been invaluable. Charles has a very open style when it comes to advising. No matter is too small, no topic is off-topic. His all-encompassing approach to advising has benefited me greatly.

I am very grateful to my thesis committee. They have been a tremendous influence on me and on this work. I would like to thank Andrea Thomaz for her many suggestions and constructive criticisms that have helped to bring the pieces of my work together. Michael Littman was pivotal in setting the direction of my thesis. Without his guidance, this would have been a very different dissertation. I would also like to thank Alex Gray. Alex always brought a "think big" attitude that challenged me to do more. Finally, my thanks to Mike Stilman whose different perspective helped me to solidify the work.

Many others have also contributed to the development of this work. Near to home, I would like to thank Arya Irani, Liam MacDermed, Jonathon Scholz, Luis Rus, David Minnen, Serge Bhat, David Roberts, John Cassel, and Kaushik Subramanian. They have been my collaborators, my colleagues, and my friends. Thank you for your help, the wonderful discussions, and your steadfast support. Further afield, I would like to thank the wonderful reinforcement learning community that has really made me feel welcomed. Special thanks to Sarah Osentoski, George Konidaris, Chris Painter, and Ron Parr. Finally, I would like to thank two early influences, Wenke Lee and Gerald DeJong, for introducing me to machine learning and reinforcement learning, respectively. Without them, I may have never entered into this field.

I could not have made it through the graduate program without the support and guidance of my friends and family. Starting with pfunk, the best research lab ever, I thank Chip Mappus, Michael Holmes, Mark Nelson, Chris Simpkins, Joshua Jones, Andrew Cantino, Christina Strong, and Peng Zhou. I thank all of the wonderful friends I made in Atlanta. They made graduate school enjoyable. Thanks to Maya Cakmak and Raffey Hamid, Santiago Ontanon, Franz and Rosa Haller, and Matt Bonner and Heather Munez. I cannot understate the importance of my long-time friends. They have given me unending support and I will always be grateful. Thanks to Hilary Nichols for her constant support and unwavering faith in my abilities. Thanks to Jeff Chang, who may live a world away but remains close in friendship. A special shout-out to my NCSSM friends! Thanks to my comrade-in-arms Jamal Rorie, Kha and Bebhinn Do, Shawn Dilda, Jarrod Johnson, Ryan Smith, Daniel Wong, Joseph Kinnarney, and August and Rachael Dwight.

Finally, I would like to thank my parents for their love and unconditional support. My accomplishments are a reflection of them. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The Markov Decision Problem (MDP) is a widely applied mathematical model useful for describing a wide array of real world decision problems ranging from navigation to scheduling to robotics. Existing methods for solving MDPs scale poorly when applied to large domains where there are many components and factors to consider.

In this dissertation, I study the use of non-tabular representations and human input as scaling techniques. I will show that the joint approach has desirable optimality and convergence guarantees, and demonstrates several orders of magnitude speedup over conventional tabular methods. Empirical studies of speedup were performed using several domains including a clone of the classic video game, Super Mario Bros. In the course of this work, I will address several issues including: how approximate representations can be used without losing convergence and optimality properties, how human input can be solicited to maximize speedup and user engagement, and how that input should be used so as to insulate against possible errors.

CHAPTER I

INTRODUCTION

Consider the task of making tea. This simple task of going to the kitchen, brewing hot water, and pouring it over tea leaves is a complex sequential decision problem. It involves navigating to the kitchen, avoiding obstacles, using the stove, and pouring hot water without spilling.

A sequential decision problem is one in which an agent must make a series of decisions in order, and actions taken have both immediate and long term effects. Our world is filled with sequential decision problems. As seen in our tea example, we are constantly solving navigation and control problems. When investing for retirement, we face the problem of what and when to buy or sell. Even games we play, such as chess or solitare, are sequential decision problems.

A Markov Decision Problem (MDP) is a mathematical formalization for describing sequential decision problems. It represents problems by breaking them up into several abstract components: a set of states the world can be in, a set of actions the agent can take, a transition function that describes how one progresses from state to state through different actions, a reward function which encodes the objective of the problem, and a discount factor which specifies the degree of preference for sooner rewards over later ones. The solution to an MDP is a *policy*, a mapping from states to actions, that maximizes the expected sum of discounted rewards.

As an example, consider a straightforward formalization of chess. The set of states would be the set of possible board positions, the set of actions would be the set of legal moves, and the transition function would describe how pieces move and the rules of capturing. Assuming tournament rules, the reward function would be an undiscounted 1 for winning, $\frac{1}{2}$ for a draw, and 0 for losing.

As a more complex example, consider the problem of trading stocks and bonds to maximize your retirement nest egg. The set of states might consist of your portfolio (*i.e.,* the set of stocks and bonds you hold) and various indicators of market conditions, the set of actions could be what you can buy or sell, and the transition function would describe how the market and your portfolio change in response to the trades you make. Assuming the goal is to maximize the return, the reward function would be the amount of money made or lost, discounted by the cost of capital.

MDPs have been studied across a variety of fields ranging from operations research [86], to decision theoretic planning [17], to control [37], to reinforcement learning [120]. The range of problems MDPs have been applied to is similarly broad:

- **Classical planning problems:** blocks world [41], freecell [147], and briefcase [42].
- **Robotic control:** simple tasks such as balancing an inverted pendulum [120], as well as more complex tasks such as riding a bicycle [106], flying a helicopter [93], and playing soccer with robots (*e.g.,* RoboCup) [118].
- **Games:** boardgames such as checkers [109], backgammon [128], chess [14], and Go [114]; video games such as Wargus [3] and Pac-man [44]; card games such as hearts [119] or poker [29].
- **Scheduling and logistics:** job-shop scheduling [151], elevator dispatching [120], crude oil transportation [22], and traffic light control [132].
- **Trading:** options pricing [137], trade optimization [24, 90], auctions [117], and mechanism design [96].

Many of these applications have led to notable successes. Tesauro's TD-Gammon algorithm [128] is the first computer program to achieve Master level play in the game of Backgammon. Ng demonstrated autonomous helicopter flight, showing an UAV's ability to handle difficult maneuvers such as inverted hovering [93]. Crites tackled the problem of elevator dispatching and achieved results surpassing modern heuristic control algorithms [28].

Despite many successes, wider application of MDPs has been hampered by challenges in reliable scaling. In the rest of this chapter, I will provide some background on methods for solving MDPs and introduce the challenges in scaling that lie therein. I will then introduce my approach for tackling these challenges, my thesis, and my contributions. I conclude this chapter with a reading guide detailing how the rest of the dissertation is organized.

1.1 Challenges in Solving MDPs

MDPs have been studied for several decades. Early solutions to MDPs include methods such as policy iteration [58] and value iteration [15]. Policy iteration iteratively improves a sequence of policies until convergence to an optimal policy. Value iteration works similarly, but focuses on the value function — a mapping from states to their long term values, which can be trivially converted to a policy. Later, Schweitzer showed that linear programming could also be used to solve MDPs [112]. These early solutions, focused on the *exact* case, are proven to converge, and are guaranteed to find the optimal policy [105].

Although these early solutions have impeccable theoretical properties, they do not scale to real world problems. Due to dependence on exact, tabular representations of the policy or value function, these methods run in time at least proportional to the number of states. Since the number of states is prohibitively large for most domains, application of these methods is generally infeasible for all but the smallest of problems. As an example, consider the number of states in chess. If a state is composed of all the pieces and their positions, then roughly estimating, there are some 64^{32} states — each piece can be anywhere on the board (64 locations) and there can be up to 32 pieces on the board (16 for white and 16 for black). This example illustrates the general principle that the number of states is exponential in the number of state elements (*i.e.,* the number of state dimensions), which can give even relatively simple domains very large state spaces.

To avoid this exponential tie, a variety of approaches have been developed. Some methods focus on specific types of MDPs and leverage the special properties of those MDPs to improve scalability. For example, some methods assume that the problem can be decomposed into a set of locally interacting components [52], or a hierarchy of subtasks [34].

Other methods assume that the MDP can be modeled using higher-level representations such as a relational representation [38]. While these techniques have had notable successes for specific types of MDPs, they cannot address the exponential tie in a general fashion.

To address MDPs with large state spaces in a general way, many have turned to *approximate* approaches. These methods trade away accuracy for scalability. Unfortunately, in doing so, they lose fundamental theoretical properties. Early work in approximation displayed a lack of basic convergence guarantees, which plagued even the simplest of domains [11, 136]. More recent results have improved the state of the art. There are now techniques that can guarantee convergence to a local optimum [100, 80]. There even approaches that can, under certain conditions, guarantee convergence to a globally optimal or near-optimal solution. Evolutionary techniques [47, 54] are one such example. However, these techniques require the design of proper mutation and crossover operators and can only guarantee convergence in the limit. Approximate linear programming methods [31, 32] are another example. Unfortunately, they require information on the relative importance of states under an optimal or near-optimal policy to be effective.

This brings us to our first challenge in solving MDPs: while results have significantly improved from where they started, it is still an open problem to find an approximate solution technique which converges to an optimal or near-optimal solution under general conditions.

Parallel research focuses on augmenting the MDP with additional knowledge to make the problem more tractable. This research direction stems from the observation that many real world problems are impractical to solve in a pure MDP formulation. Indeed, even if better approximation methods were developed, some MDPs are intrinsically complex and may require more samples than practical. A pure MDP formulation also attempts to solve the decision problem from scratch — an approach that is rarely taken in human and animal problem solving.

A very promising direction toward this end is the use of human input for obtaining additional knowledge to augment the MDP model. This direction includes work in learning by demonstration [6], imitation learning [4], apprenticeship learning [2], learning through advice [5], and reward shaping [106]. Leveraging human input has achieved some impressive

results. These include helicopter flying [93], biped locomotion [89], teaching a AIBO robot basic soccer skills [51], and many others. Advocates of this line of research also point out that human help is often readily available and cheaply obtained. Indeed, consider common applications such as robotic assistants in homes and administrative assistants on computers. In these and other applications, humans are the end-users — making them a readily available and well motivated source of additional knowledge.

Using human input is not without its difficulties. One must be mindful of the target audience. Our focus will be on the general population, *i.e.,* end-users. The unique characteristics of this audience should be carefully considered. Users, for example, generally know little of machine learning or automated planning. This unfamiliarity poses certain difficulties when designing interaction mechanisms between users and machines. For example, one cannot ask users directly for the set of relevant features because they will typically be unfamiliar with the (technical) definition of "features". Users also have limited time and can be prone to making mistakes. In short, while users can be a rich source of helpful information, we cannot expect them to act as an oracle. Rather, they are better characterized as limited sources of noisy, but rich information.

Early works on using human input [108, 73, 111, 1] focused primarily on fundamental methods and tended to target highly trained audiences. These early works commonly assume the existence of an optimal or near-optimal oracle, use researchers themselves as the source of human input, or bring in outside domain experts to provide high quality input. As a result, the nuances of dealing with end-users have been largely ignored. The methods developed in early works can often be difficult to apply to end-users, and tend to be sensitive to errors in the input. Consideration of users has only recently begun to be emphasized [61, 129, 131, 66].

This discussion brings us to a second challenge in solving MDPs: how to leverage human input to help solve MDPs faster. Note that by human input, I refer specifically to input from the general population, *i.e.,* input from end-users.

1.2 Thesis

I care about solving large MDPs, *i.e.,* MDPs with large state spaces, and solving them in practice.

In this dissertation, I take a two-pronged approach:

A) I develop a method for approximately solving general MDPs that is guaranteed to converge to an optimal or near-optimal policy.

B) I explore how to leverage human input. Specifically, I tackle two issues, (1) how to leverage human input while insulating against possible errors, *i.e.,* how to *safely use* human help, and (2) how to solicit human input, *i.e.,* how to *ask for* help.

Finally, I will show how to combine these two approaches to build a joint system for solving general MDPs that can offer significant speedups in practice, while still maintaining convergence and optimality guarantees. I claim that:

> *Non-tabular representations in conjunction with human input can approximately solve MDPs with global optimality and convergence guarantees in complexity non-dependent on the size of the state space, yielding several orders of magnitude speedup over tabular methods.*

By non-dependence on state space size, I mean that the complexity depends upon characteristics such as the depth of the problem that may be influenced by state space size, but not on state space size directly.

To address approximation in MDPs, my work relies on the insight that a large portion of the difficulties in solving MDPs lies in the prevalent use of bootstrapping: the practice of estimating values or actions on the basis of other estimates, which themselves are most likely incomplete and in need of correction. Further, bootstrapping is typically used with *arbitrary* initial estimates. This situation creates what I call an iterative refinement process, in which the MDP is solved by repeated corrections of an almost surely inaccurate initial guess. This process has several consequences. First, it creates inefficiencies — computations based on arbitrary estimates will almost certain yield incorrect solutions that will only need to be recomputed at a later time. Second, it generates unpredictable intermediate results.

Finally, it makes convergence difficult under approximation — errors from approximation may overwhelm any improvements from refinement, leading to non-convergence or even divergence.

My work eschews this iterative refinement process, opting instead to use an expanding approach. The fundamental idea of this approach is to compute the optimal solution for some initial set of states directly, and to then expand this set, building outward by leveraging previously computed solutions. Error propagation is much easier to reason about in this framework and intermediate results are readily understandable. To obtain the initial set of states, we leverage the observation that every MDP has at least one terminal state[1], a state that the system must eventually enter, and which allows no escape once entered. Terminal states have closed form solutions which can be easily computed. Expanding the set of solved states poses a greater challenge. The solution to an unknown state, a state outside the solved region, may depend upon other unknown states. In the general case, it is possible for unknown states to be interconnected such that solving any one state would require the solving of all states, creating a deadlock situation.

One way to address this difficulty is to compromise and use a worst case estimate for unknown states. This move introduces a limited version of bootstrapping, used only to generate expansions, enabling us to break the deadlock and still keep some of the advantages of the expansion approach. Following this line of reasoning, we develop a variation of value iteration for the exact case that is often orders of magnitude faster and with worst-case complexity that is no worse than standard value iteration. For the approximate case, this line of exploration leads to some earlier work by Boyan [19, 18]. Unfortunately, positive results are limited — even the limited form of bootstrapping can cause difficulties. Assuming an adequate approximate representation, we *can* guarantee convergence. However, (1) we can only do so in deterministic or acyclic domains, and (2) we can only guarantee convergence to a policy, not necessarily an optimal policy.

An alternative to addressing this difficulty is to use sampling planners [64, 68], which compute, for an individual state, an estimate of the state's optimal value or action. Through

[1] Any MDP without a terminal state can be trivially converted into one that does [99].

repeated application of these planners on multiple unknown states, we can generate expansions of the solved region. Extending Boyan's previous work with this technique, I prove convergence and derive tight bounds on optimality. However, use of sampling planners is not without its costs. It trades an exponential dependence on the dimensionality of the state space for, in the worst case, an exponential dependence on the depth of the problem. Fortunately, there is some hope for alleviating this exponential dependence. First, most practical applications do not belong to this worst case; they are typically only exponential in the depth of the expansion size. Second, we can leverage significant existing works in classical and decision theoretic planning that have been grappling with the exponential nature of planning for decades. Third, the most successful methods for both large and deep problems such as Go are based on sampling planners and show compelling effectiveness [46, 43]. In empirical studies, I demonstrate that this alternative formation can yield significant practical performance improvements (see Chapter 3).

To address the second prong, the use of human input, I focus on two aspects: how to leverage human input so as to insulating against possible errors, and how to solicit human input. For the latter aspect, we turn to a paradigm known as socially-guided machine learning (SGML) [131]. This paradigm is motivated by the observation that while users are generally not experts in machine learning, and typically have never taught a machine, they are experts in social teaching (*e.g.,* tutelage, imitation). Thus, to make teaching natural for users, one should tap into these existing social teaching skills. Following this line of reasoning, I study the importance of interactivity, known to play an important role in human teaching, to the solicitation of human input. Our user study finds that interactivity not only improves the learner's (*i.e.,* the machine's) performance but also improves the user's experience.

For the former aspect on insulating against errors, we turn to indirect uses. Direct use of human input as examples of appropriate behavior cannot guard against errors. If suboptimal inputs are given, the system, being unable to distinguish errors, will imitate it and generate similarly suboptimal results. My approach to this problem is to extract supportive knowledge from human input, *i.e.,* to use human input as *scaffolding.* Scaffolding

is a concept from educational psychology in which the teacher plays the supportive role of enabling the learner to achieve something they would not be able to accomplish otherwise.

I explore two methods to this end. In the first method, we use demonstrations to discover hierarchical decompositions so that the machine, *i.e.,* the learner, can solve the problem in a timely fashion. Decompositions are known to be able to provide significant speedup [34, 62], and using human input in this fashion allows us to avoid direct imitation and its accompanying errors. Empirical experiments, including user studies, confirm the effectiveness of the approach. We observed speedups of up to 30 fold, high tolerance for errors in the demonstrations, and learned policies that regularly outperform their human demonstrators.

In the second method, we will use demonstrations to identify relevant features. Again, by avoiding direct imitation and using demonstrations for supportive knowledge, we hope to obtain performance benefits while being robust to errors in the input. Empirical experiments, including user studies, confirm the effectiveness of the approach. We observed speedups up to several orders of magnitude, higher sample efficiency as compared with direct learning from demonstration, and learned policies that also outperform their human demonstrators.

Finally, to combine my two-pronged approach, I develop an interaction scheme called training regimens which, applied carefully, enables the use of human input while maintaining the theoretical properties of the approximation algorithm. Training regimens also take advantage of decompositions but of a different kind. It is based on the insight that solutions to states further away from the goal (*i.e.,* a terminal state) often leverage the solutions of states closer to the goal. In essence, training regimens provide a decomposition of the state space from "short" to "long" problems by controlling what states to solve and the order in which to solve them. In empirical results, the combined system shows several orders of magnitude speedup over tabular methods, and is competitive with other approximate solution methods.

I summarize the contributions of this dissertation below:

- A replacement for the standard value iteration algorithm which often yields speedups

of several orders of magnitude while having worst-case complexity no worse than that of value iteration.

- An algorithm for approximately solving general MDPs that has complexity non-dependent on the size of the state space, is guaranteed to converge, and whose result can be bounded to the global optimum. The algorithm can make use of any approximation scheme that can provide confidence intervals on its estimates.

- User studies that show the importance of interactivity when soliciting help from human users. Interactivity improves system performance by giving users a better understanding of the learner, enabling them to adapt their teaching strategies. Interactivity also improves the user experience, increasing the level of engagement felt.

- Two algorithms that leverage human input to obtain significant speedup while simultaneously insulating against possible errors within that input. The first utilizes demonstrations to induce problem decompositions, while the second utilizes demonstrations for feature selection.

- A human interaction scheme, training regimens, which can pair with our function approximation algorithm to provide speedup from human input without affecting correctness.

1.3 Reading Guide

The layout of this dissertation begins with a chapter on the relevant background in which I introduce our notation and provide an overview of related works. The next three chapters represent the primary content where I present the bulk of my work along with any additional context specific to the work. In the final chapter, I conclude with a summary and discussion of future work.

The main content of the dissertation, presented in Chapters 3, 4 and 5, is organized around our two-pronged approach, and follows the same order. The first two chapters address the use of approximation in MDPs, and then the use of human input. Each of these chapters, follows the same basic structure. I first provide some motivation and background,

which will then allow me to state the main issue/problem to be resolved. The rest of the chapter is then dedicated to the solution(s).

To justify the thesis, I must prove certain theoretical guarantees as well as show significant speedup over tabular methods. Most of the work in demonstrating optimality and convergence guarantees will be Chapter 3. Demonstrations of speedup are split into two places. Some empirical results are presented in Chapter 3 along with the theory. Most of these results focus on domains small enough to perform direct comparisons with tabular methods. The rest of the empirical results are presented in Chapter 5 which covers how to combine human input with approximations. These results focus on larger domains, including a clone of the video game, Super Mario Bros.

CHAPTER II

BACKGROUND AND RELATED WORK

In this chapter I introduce the definitions and notation I will be using throughout the dissertation. I then provide an overview of related work with which to contextualize this dissertation.

We begin with the definition of the Markov Decision Problem (MDP). An MDP is defined by the tuple (S, A, T, R, γ) where:

- S is a set of states
- A is a set of actions
- $T : S \times A \to \prod(S)$ is a transition function in which $T(s, a)$ specifies the next state distribution for taking action a in state s. In other words, $T(s, a)$ is a random variable specifying the next state. At times I will use $T(s, a)(s') \to [0, 1]$ to denote the probability of reaching state s' when taking action a from state s.
- $R : S \times A \to \mathbb{R}$ is a reward function in which $R(s, a)$ specifies the immediate reward received when taking action a in state s.
- $\gamma \in [0, 1)$ is a discount factor.

The solution to an MDP is a *policy*, $\pi : S \to A$, which maps states to actions prescribing what action the agent should perform in a given state. In particular, the solution to an MDP is the *optimal* policy, denoted π^*, which maximizes the expected sum of discounted rewards: $\sum_{i=1}^{\infty} \gamma^{i-1} R_i$ where R_i is the reward received when taking the i^{th} action.

Our definition of the Markov Decision Problem is related to the well known Markov Decision Process [105]. The latter is typically defined by the four tuple (S, A, T, R) with the same semantics as the former. The Markov Decision Process, however, is not a fully defined problem as it lacks an optimality criterion. Some of the optimality criterions that

have been studied with Markov Decision Processes include maximizing average reward, finite sum of rewards, and infinite sum of discounted rewards [99]. It is this last optimality criterion that I focus on and use to define the Markov Decision Problem.

While there are no requirements on how a solution policy must be represented, I make the non-functional requirement that a solution policy must be computable quickly — within the real-time constraints of the problem it models. Without such a requirement, the definition of the MDP itself would suffice as a solution. While the exact real-time constraints are problem dependent, as a rule of thumb, we expect policy computation to take no more than a few seconds for any single state.

We now define some concepts useful for discussing MDPs.

An absorbing state is one from which the system can never escape, once it has been entered. That is, if a state s is absorbing, the transition function from s for any action always leads back to s. A terminal state is an absorbing state which the system must eventually enter. Every MDPs can be trivially converted so that it has at least one terminal state.

A *mixed* or *stochastic* policy, $\pi : S \to \prod(A)$, is a generalization of policies that maps states to a distribution over actions. In other words, $\pi(s)$ is a random variable specifying the action to take in state s. We use $\pi(s)(a) \to [0, 1]$ to denote the probability of taking action a in state s. At times, we may refer to a policy as deterministic or fixed, to contrast it from mixed polices.

A value function, $V^\pi : S \to \mathbb{R}$, maps states to their utilities or *values* under policy π. The value of a state s under a policy π, $V^\pi(s)$, is the expected sum of discounted reward an agent receives when following policy π from state s. The optimal value of a state, $V^{\pi^*}(s)$ or just $V^*(s)$ is the value of state s when the optimal policy is followed.

A Q or action value function, $Q^\pi : S \times A \to \mathbb{R}$, maps state-action pairs to their Q-values under policy π. The Q-value of a state-action pair under policy π is the expected sum of discounted reward for taking action a in state s and following policy π from there on. The optimal Q-value of a state-action pair, $Q^{\pi^*}(s, a)$ or just $Q^*(s, a)$, is the Q-value of the state-action pair under the optimal policy.

The value function and the Q function are related:

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s)) \tag{1}$$

$$Q^{\pi}(s, a) = R(s, a) + \gamma \mathbb{E}[V^{\pi}(T(s, a))] \tag{2}$$

The optimal policy, value function, and Q function have additional relations.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a) \tag{3}$$

$$V^*(s) = \max_a Q^*(s, a) \tag{4}$$

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}[V^*(T(s, a))] \tag{5}$$

We can expand the latter two relations to derive the Bellman optimality equations.

$$V^*(s) = \max_a [R(s, a) + \gamma \mathbb{E}[V^*(T(s, a))]] \tag{6}$$

$$Q^*(s, a) = R(s, a) + \gamma \mathbb{E}[\max_{a'} Q^*(T(s, a), a')] \tag{7}$$

The Bellman optimality equations also define the core of many elementary solution techniques. Value iteration, for example, uses the Bellman optimality equation as an update rule:

$$V^{k+1}(s) = \max_a [R(s, a) + \gamma \mathbb{E}[V^k(T(s, a))]] \tag{8}$$

where $k \in \mathbb{Z}^+$. For conciseness, it will be useful to define this update rule as a nonlinear vector operator, P, such that $V^{k+1} = PV^k$. The magnitude of the update $|V - PV|$ is known as the *Bellman error*. The Bellman operator, P, can be viewed as a one-step lookahead improvement operation.

It is well known that P is a contraction mapping in maxnorm with contraction coefficient $\gamma : ||PV - PV'||_\infty \leq \gamma ||V - V'||_\infty$. Further, P has a unique fixed point at V^* — where the Bellman optimality equations hold [105]. As a result, for arbitrary V^0, the sequence $V^0, V^1 = PV^0, V^2 = P^2V^0, \ldots, V^k = P^kV^0$ is guaranteed to converge to V^*.

Value iteration works by exploiting this fact, performing full sweeps of the state space every iteration, updating each state according to this operation. When an iteration of updates yields no change to the value function, the fixed point has been reached and the

Require: $M = (S, A, T, R, \gamma)$, ϵ
// M - the MDP to solve
// ϵ - the precision parameter

Create initial value function V arbitrarily, *e.g.,* $V(s) = 0, \forall s \in S$
repeat
$\Delta = 0$
for all $s \in S$ **do**
$oldv = V(s)$
$V(s) = \max_a(R(s,a) + \gamma \mathbb{E}[V(T(s,a))])$
$\Delta = \max(\Delta, |oldv - V(s)|$
end for
until $\Delta \leq \epsilon$
return V

Figure 1: Basic Value Iteration

algorithm halts. In practice, value iteration halts when the maximum change in an iteration drops below some precision threshold ϵ. The value function resulting from this stopping rule can be bound to be within $\frac{\epsilon}{1-\gamma}$ of the optimal V^* [105]. Figure 1 shows the standard value iteration algorithm.

At times it will be more convenient to think in terms of cost, $G(s,a) = -R(s,a)$, rather than rewards. In these cases, the *cost-to-go* of a state $J(s) = -V(s)$, refers to the negation of its value. An optimal policy of an MDP is one which maximizes V, the expected utility, or equivalently, minimizes J, the expected cost-to-go.

A *trajectory* through an MDP is a sequence of transitions from state to state, sampled by following a policy π from some initial state s_0. The *return* of a trajectory is the discounted sum of rewards encountered along the trajectory. The process of sampling many trajectories is sometimes referred to as *policy rollouts*.

Averaging the return of many sample trajectories allows one to compute the *return* of a policy: $f(\pi) = \mathbb{E}[V^\pi(x_0)]$ where x_0 is an initial state sampled from some start state distribution. The expected return of a policy measures the quality of a policy *wrt.* a start state distribution. A policy that obtains the maximum return *wrt.* a state state distribution that provides nonzero measure over all states, is an optimal policy.

MDPs can be considered to have an effective finite *horizon* — the number of steps into the future to consider. Although the model formally calls for maximizing the infinite sum

of discounted rewards (and thus to look infinitely far into the future), in practice, MDPs are solved to some precision level which places an effective limit on the horizon. If nothing else, floating point precision limitations impose a practical precision level. For a particular precision level, rewards after a certain point are negligible as they are so heavily discounted that the sum of their impact falls below the precision. This point is the effective horizon of the MDP. If we assume, without loss of generality, that the range of the reward function falls within $[0, 1]$, then given precision ϵ, the effective horizon of the MDP is $\lceil \log_\gamma(\epsilon(1-\gamma)) \rceil$. "Horizon" is sometimes referred to as "depth" in the literature, and we will also make use of it here.

MDPs are P-complete[1] [95, 78]. Specifically they are polynomial in $|S|, |A|$, and B, the number of bits needed to express the transitions, rewards, and discount factor of the MDP as rational numbers. It is the dependence on $|S|$ that most often poses difficulties for application to real world problems.

In the rest of this chapter, I will provide an overview of some related work. An understanding of the various research threads being explored and their history is necessary to understand how the focus and contributions of this dissertations fits into the larger research narrative, and the larger implications of the work. Unfortunately, there are too many different approaches and too many facets to consider for one picture or one view point to concisely cover the field. Instead, in the following sections, I will cover related work and how my work fits into that context from a series of different view points. It is my hope that with these different view points in mind, the reader will be able to construct a deeper understanding of the context of my work. The first section will organize research by the fundamental representation and basic solution strategy used. The next section will organize research by the approach taken to deal with large state spaces. In the final section, I will broaden the scope and consider works that solve similar and related problems to provide a larger perspective.

[1] A decision problem is in P if it can be decided in polynomial time. It is P-complete if it can be decided in polynomial time and every problem in P can be reduced to it.

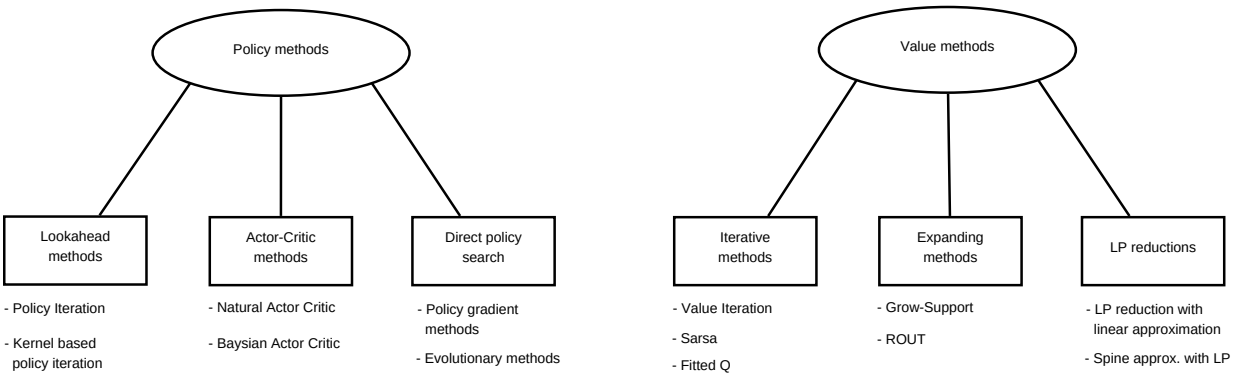

Figure 2: Organization of fundamental approaches to solving MDPs. There are two broad categories based on representation: policy methods and value methods. Each category is further subdivided into groups based on the solution approach used. For each group, a few representative algorithms are given as examples.

2.1 Fundamental Approaches to Solving MDPs

One way to characterize the myriad of solution methods to MDPs is by their choice of solution representation. I organize them into two broad categories, **value-based methods** and **policy-based methods**. Value based methods have solution representations based primarily around the value function (either V or Q). Policy-based methods do not; they typically rely on some alternative, explicit policy representation. We can further divide methods within each category by the basic solution approach used. A summary of the organizational structure I impose is shown in Figure 2. The contributions presented in this dissertation are all value-based methods. Some are based on iterative approaches, but the bulk of the work, particular the portions involved in proving tight error bounds, fall under the expanding approach.

Policy-based methods hold the policy as the central solution representation. This has a few advantages. First, computing the proper action for a given state is direct — unlike value methods, no comparisons over different actions need to be made. As a result, the process tends to be faster. Second, by representing the policy directly, these methods can learn explicitly stochastic policies. Stochastic policies are often easier to work with than deterministic policies because they are smoother — small changes to the policy result in small changes to the return of the policy. Policy methods can be subdivided into three groups.

Require: $M = (S, A, T, R, \gamma)$, ϵ
// M - the MDP to solve
// ϵ - precision parameter

1: Create initial policy π arbitrarily
2: **repeat**
3: $\Delta = false$
4: $V =$ `pieval`(M, π, ϵ)
5: **for all** $s \in S$ **do**
6: $olda = \pi(s)$
7: $\pi(s) = \underset{a}{\text{argmax}}(R(s,a) + \gamma \mathbb{E}[V(T(s,a))])$
8: $\Delta = (olda == \pi(s))$
9: **end for**
10: **until** $\not\Delta$
11: **return** V

Figure 3: Classic Policy Iteration. `pieval` is a subroutine which performs policy evaluation (*e.g.,* see Figure 4)

Lookahead methods are a group of policy methods that generate a sequence of improved policies by computing the value function of the current policy (*i.e.,* V^π or Q^π) and then using a greedy k-step lookahead search on top of this value function to create the next policy. The process of computing the value function of a particular policy is known as *policy evaluation.* One step lookaheads (*i.e.,* $k = 1$) are the most common.

The paragon of this group of methods is the classic policy iteration (PI) algorithm (see Figure 3. In PI, some arbitrary policy is chosen initially and then iterative improvement is performed until no further improvements can be made. In every iteration of PI, V^π is computed using a policy evaluation subroutine (line 4) based on value iteration (see Figure 4). The resulting value function is used to perform one-step lookahead to create the improved policy (line 7).

Examples of work in this group include kernel based policy iteration [146], rollout sampling approximate policy iteration [35], and many others [70, 41, 75]. The policy representation used in these methods vary widely, ranging from relational representations, to functional forms (*e.g.,* linear models), to decision lists. The method of policy evaluation also varies with some methods using policy rollouts. What all these methods share in common is the use of lookahead search on top of a value function to compute the improved policy.

Require: $M = (S, A, T, R, \gamma)$, π, ϵ
  // M - the MDP to evaluate our policy in
  // π - the policy to evaluate
  // ϵ - precision parameter

  Create initial value function V arbitrarily
  repeat
    $\Delta = 0$
    for all $s \in S$ **do**
      $oldv = V(s)$
      $V(s) = R(s, \pi(s)) + \gamma \mathbb{E}[V(T(s, \pi(s)))]$
      $\Delta = (oldv == V(s))$
    end for
  until $\Delta \leq \epsilon$
  return V

Figure 4: Policy Evaluation using value iteration

Some techniques are not considered to belong in this group although they share a similar name. For example, methods like least squares policy iteration (LSPI) [74] and representation policy iteration [82] are not considered to belong in this group. These methods are fundamentally based on the Q function rather than an explicit policy representation, and so are considered value-based methods.

Actor-critic methods are a group of policy methods that find improved policies by using the value function of the current policy as a source of gradient-like information [72]. They are called actor-critic methods because they have a policy representation (the "actor"), and simultaneously, a separate value function representation (the "critic"). The most common methods in this group use the temporal difference (TD) error of the value function to compute the appropriate gradient direction [145, 12, 141, 120, 16]. TD error, which is computed at every step taken, represents how things have gone better or worse than expected. In implementation, these methods keep an online estimate of the value of the policy being executed, *e.g.,* V^π. The TD error after taking step t can then be computed as $R_t + \gamma V(s_{t+1}) - V(s_t)$.

"Direct" policy search methods are a group of policy methods conceptually different from the other two. These methods are not based around any concept of computing a value function to construct sequences of improved policies. Instead, they cast the problem of finding an optimal or near optimal policy as a general optimization problem. In this

formulation, the set of policies is the feasible set, and the objective function to maximize is the expected return of the policy. Casting the MDP as an optimization problem presents certain advantages. It lets us leverage existing techniques in the field of optimization, and allows us to specify the set of policies to search. On the other hand, specifying an intelligent set of policies to search can be difficult if domain knowledge is not available. Casting the MDP into a optimization framework also brings its own set of difficulties. The optimization problem may be non-linear, non-convex, and/or non-differentiable. Evaluation of the objective function may also be expensive depending on the horizon of the problem.

Regardless of the pros and cons, this paradigm has led to a host of different techniques. Policy gradient methods assume a differentiable policy representation and use the gradient to find local optimas [141, 92, 100, 125, 69]. Later methods in this line of research explored the use of natural gradients [63, 101, 16]. Evolutionary techniques [47, 59, 54] search over populations of policies and improve on a generational basis. Improved policies are constructed through mutation and crossover operations. Typically, some form of policy evaluation serves as the fitness function. Evolutionary methods have been shown to be competitive [55] and have had success in practical applications such as fin-less rocket control [48]. Cross entropy methods [83, 20, 21] treat the policy as a method for generating random trajectories, and construct improved policies on the basis of the best (sometimes referred to as elite) trajectories sampled. A notable success of this approach is in Tetris where the policy learned out scored previous attempts by almost two orders of magnitude [126]. Still other methods in this group include those based on simulated annealing [10, 76], and those that approach the optimization problem as inference problem [133, 143].

Value-based methods compose the other broad category of MDP solution techniques. This category of methods is differentiated from the previous policy-based one by the choice of the value function as the primary representation. A variety of solution approaches belong in this category.

Before discussing various value-based methods, it is useful to note that the value function can be well defined as a system of equations, with $|S|$ unknowns and $|S|$ equations. For each state, the unknown is the value of that state, $V^*(s)$, and the equation is the Bellman

optimality equation for that state, $V^*(s) = \max_a[R(s,a) + \gamma\mathbb{E}[V^*(T(s,a))]]$. The same holds similarly for the Q function, but for state-action pairs rather than for states. The difficulty in solving the value function lies in the fact that the equations are nonlinear and non-differentiable due to the max operation.

Iterative methods are a group of value methods based around iterative refinement of the value function until convergence. Much like policy methods, these techniques start with some initial, usually arbitrarily, value function which is iteratively improved. This group of methods is one of the earliest explored, and one of the most well studied. Many of the best known algorithms in reinforcement learning fall in this group. Examples include value iteration [105], real time dynamic programming (RTDP) [13], Q-learning [140], and Sarsa [107, 123].

Methods in this group typically compute improved values via some form of lookahead. The most common form of lookahead, the single step version, is sometimes referred to as a "backup". The "backup" operation is based on the Bellman optimality equation:

$$V^{k+1}(s) = \max_a[R(s,a) + \gamma\mathbb{E}[V^k(T(s,a))]] \tag{9}$$

$$Q^{k+1}(s,a) = \max_a[R(s,a) + \gamma\mathbb{E}[\max_{a'} Q^k(T(s,a),a')]] \tag{10}$$

Iterative methods can be differentiated by the details of how improvement is performed. In particular, by (1) whether the value function is improved for all states (*i.e.,* full sweep) or more focused on some subset (*i.e.,* partial sweep), (2) whether all next states are considered (*i.e.,* full backup) when computing improved values or just a sample of the next state (*i.e.,* sample backup), and (3) how deep of a lookahead is used when computing improved values (*i.e.,* shallow vs deep backups). In these terms, value iteration is a full sweep, full backup, and shallow backup method. RTDP is similar, but based on partial sweeps. By contrast, Sarsa and Q-Learning are both partial sweep, sample backup, and shallow backup methods. Partial sweeps can be further refined, for example, by whether or not the sweeps are on-policy — when the states to be improved are generated by following the current policy. In this parlance, the difference between Sarsa and Q-Learning is that the former is on-policy, while the latter is off-policy. While further distinctions can be made, a full treatment is

beyond the scope of this dissertation.

Some techniques within this group have names suggesting otherwise. Examples include the aforementioned LSPI [74] and representation policy iteration [82]. These methods generate a sequence of improved Q functions and compute improvements on the basis of lookahead just like other methods in the group. They are sample backup, and shallow backup techniques. Because they leave the set of state-action pairs to update as a parameter of the algorithm, they can be either full sweep or partial sweep algorithms.

Expanding methods are a group of value methods [19, 18, 148] based on direct computation of a partial value function, and then expanding that value function until it is full — until it covers all states. In other words, these methods compute the exact solution for some initial set of states (typically the set of absorbing states as they have closed form solutions), and then use those solutions to compute solution to other unknown states thereby expanding the solved set. A key differentiator between these methods and the iterative methods is that expanding methods do not bootstrap. These methods only compute solutions on the basis of other solved states, never on the basis of incomplete solution estimates. The expanding method has received relatively little attention over the years. However, as I show in this dissertation, it is key to enabling the use of approximation in a safe, and controllable manner.

Linear programming methods are a group of value methods that solve the value function by reduction to a linear programming (LP) problem [112, 134, 31, 32]. The resulting LP has $|S|$ variables and $|S||A|$ constraints. Due to the high number of variables and constraints, the LP formulation is not of particular use for solving the exact case. The approximate case is much more interesting. Using a linear approximation, the LP formulation can solve the value problem with error bounds that can characterize the quality of the approximation *wrt.* the optimal value function. Unfortunately, the algorithm requires additional information such as the relative importance of states. This information is critical in controlling how the LP solution spreads out approximation errors, and without it, the bounds become very loose.

2.2 Strategies for Scaling Solutions to MDPs

One of the main difficulties in solving MDPs is dealing with large state spaces. In this section I will consider related work organized by the strategy used to scale to large state spaces. As a result, I will not be addressing traditional methods such as value iteration or policy iteration. These are considered in the previous section. I organize the field of works into three main categories: (1) exact methods which leverage richer representations and specialized optimization, (2) approximate methods that trade away accuracy for improved scalability, and (3) extra-MDP methods that leverage additional knowledge outside the MDP model to obtain scalability. Most of the contributions of this dissertation will fall under the last two categories. Specifically, the first prong on approximation belongs to the category of approximate methods, and the second prong on leveraging human help will belong under extra-MDP methods.

Exact methods are characterized by their use of exact representations and their pursuit of exact solutions. As a result, they can typically guarantee the same desirable theoretical properties as early solutions such as value iteration. Exact methods obtain scalability by focusing on specific types of MDPs which have properties that enable certain optimizations. By taking advantage of these optimizations, they are able to make impressive gains towards scalability.

Research on factored MDPs, for example, assume that the problem can be decomposed into a set of locally interacting components. In particular, they assume that the state is represented by a set of random variables, that the transition function is defined by a dynamic Bayesian network [33], and that the reward function is factored additively into a set of localized reward functions, each of which only depends on a small set of variables. By taking advantage of these assumptions, researchers have solved problems with over 10^{40} states [52].

In similar work, hierarchical methods assume that the problem can be decomposed into a set of subtasks, each of which needs only a small set of states or state variables to solve. A variety of hierarchical methods exist [34, 98, 122]. Some are more free form, allowing primitive actions and subtask solutions to freely mix, while others are more structured to

take more direct advantage of the hierarchy. A downside of early work in hierarchical methods is the assumption that the hierarchy is given. Follow up work has focused significant attention on automatic induction of problem decompositions [85, 84, 56, 62].

Other methods take a slightly different tack, focusing on leveraging richer representations. For example, work in relational reinforcement learning [38] assumes a relational representation of states which enable the authors to reduce the effective size of the state space and obtain significant efficiencies. Methods, such as SPUDD [57] focus on leveraging richer representations for the value function and show that computation in the richer, lifted space avoids enumerating the state space and results in significant speedups.

Overall, advances in this category have made significant improvements to scalability — but for various special types of MDPs. By their very nature of exploiting special properties, they limit their applicability. The next group of methods focus on increasing scalability for the general case.

Approximate methods are characterized by trading away accuracy for scalability. Unlike exact methods, they typically focus on general MDPs. However, by trading away exactness, they lose many of the theoretical guarantees enjoyed by exact approaches. Early work in approximate methods displayed a lack of basic convergence guarantees which plagued even the simplest of domains [11, 136, 87].

Follow up work developed theory focused on explaining divergence and showed that certain types of approximators, namely those that are non-expansive in the max norm, are safe and will converge [49, 50]. These, so called "averagers", include approximation schemes such as local weighted averaging or linear interpolation but not models such as (local) linear regression or neural networks. Averagers are distinguished in that they compute the value of a given state as a linear combination of the known values of nearby states. Further, being non-parametric methods, in the limit of infinite samples, averagers can yield approximate solutions arbitrarily close to the optimum.

More recently, a host of results have emerged giving us theory for a broader set approximators. Under the umbrella of policy oriented methods, we have policy gradient methods [125, 100] that are guaranteed to converge to a local optima assuming a differentiable

policy representation. In the same umbrella, we also have evolutionary techniques [54] which are guaranteed to converge to the reachable global optimum in the limit. These techniques can be used with a broad range of approximation models as long as proper mutation and crossover operators are given.

In value function based approaches [45], we have LSPI [74] which addresses approximation under linear models. LSPI can guarantee convergence to a region and has yielded many practical successes []. Owing in part to these successes and in part to its simplicity, LSPI has been a staple of function approximation approaches. Recently, we have also seen advances in approximation for TD methods [124, 80]. These advances offer, for the first time, convergence for TD methods. Specifically, they guarantee convergence to a local optimal for linear models, and in later work, for any smooth approximator.

Finally in linear programming based approaches, we have recent results [31] that can provide tight bounds *wrt.* the global optimum for linear models. Unfortunately, the method requires information on the relative importance of states to control how the LP solution spreads out approximation errors; without it, the bounds become very loose.

Overall, while initially lacking in convergence and optimality guarantees, recent results have made significant improvements. For approximators that are non-expansive in the max norm (*i.e.,* "averagers") the results are good – convergence is guaranteed, and the error of the approximation can be made arbitrarily small *wrt.* the global optimum given sufficient samples. For other approximators, ranging from support vector regression to local linear regression, the story is mixed. For smooth approximators, the best guarantee is convergence to a local optima [81]. Surprisingly, for the basic linear model, the best guarantee is also convergence to a local optima. Certain techniques such as the approximate LP method can offer bounds *wrt.* the global optimum but they are too loose to be useful unless the algorithm is given information on how to spread out approximation errors. Addressing this deficiency comprises the first major portion of this dissertation. The solution we present and the bounds we develop can guarantee convergence *wrt.* the global optimum, for any estimator providing confidence intervals including linear regression and local linear regression.

Extra-MDP methods focus on augmenting the MDP with additional information to

improve scalability. Methods in this group are varied. Approaches such as learning from demonstration [6] and imitation learning [104] rely the availability of examples of appropriate behavior. Learning from advice [5] relies on a source of critiques and suggestions to speed up learning. Reward shaping [106] methods focus on obtaining performance improvements by augmenting the reward function to reduce sparsity and guide exploration. Transfer learning methods [127] focus on using the learned internal knowledge from (previously) solving similar problems to obtain improvements on solving the current problem. Even some specialization methods can be considered extra-MDP methods as some make assumptions needing additional information. For example, many early hierarchical decomposition methods assumed that the hierarchy is given as an additional input.

A key issue for extra-MDP methods is the source of additional information. Some methods [104] assume a multiagent environment where other agents become a natural source of additional information. Other methods [102] assume an extended or life-long learning scenario where previous problems and their solutions can be the source. A third, very popular direction is to use human input as the source.

A myriad of methods fall under this last direction. Use of human input can be characterized by the type of humans the method expects to leverage. Some methods target humans who both understand the system as well as the domain, *e.g.,* the researchers themselves. For example, some decompositional techniques [34] expect the hierarchical breakdown of the task to be given *a priori*. Other methods target a broader audience, domain experts. For example, many learning by demonstration methods expect the human demonstrator to provide optimal or near-optimal solutions [93]. More recently, methods [61, 129] have begun to target the general audience — end-users. While this audience is the most common, it also comes with its own set of difficulties. For example, end-users have limited patience, they generally do not know how to teach machine learning programs, and they often give suboptimal or erroneous solutions. Work towards addressing these difficulties is still sparse but becoming more widely studied [131, 67, 148]. It is also the second major focus of this dissertation. Specifically, I explore issues in how to insulate against errors and suboptimalities in human input, and how to solicit that input. Much of the work on structuring

solicitation builds upon previous work in [131]. Work on insulating against input suboptimalities is more independent. This area is more unexplored and prior work is spare, the closest recent work being [67].

2.3 A Broader Perspective

In this dissertation we focus on solving the MDP. There are several related problems which will be useful to consider, to place the MDP in a broader context.

We first consider the state-specific Markov Decision Problem. In the standard MDP, the problem is defined by the tuple $M = (S, A, T, R, \gamma)$, and the solution is a policy mapping states to actions. The state-specific MDP is a narrower version of the problem which targets a specific state. Formally, it is defined by the MDP M and a query state s_q. The solution to this problem is the action that should be taken in s_q to maximize the sum of discounted rewards. Clearly, the solution to an MDP M, will suffice as the solution for any state-specific problem in M. As a result, all of the work for solving MDPs that we discuss in this dissertation can be applied to this more narrow formulation. The reverse is also possible. A solution method for solving a state specific MDP can generate a solution to the broader MDP through repeated application for every state. If this approach is taken offline, it transforms the MDP into a supervised learning problem. If this approach is taken online, and the solution method is fast enough to meet the real-time constraints of the MDP, it suffices as the solution by itself. For example, many of the best solutions to the game of Go are state-specific MDP solvers because Go is a turn-based game and players typically have at least a few minutes to make a move.

Due to the close relationship between the state-specific MDP and the general MDP, it will be useful to consider solution methods for this narrower problem. Specifically, we focus on techniques that take advantage of the narrower definition. The primary benefit of this narrow focus is that it limits the set of states that must be considered to just those that are reachable[2] from the query state. MDP solutions that perform partial sweeps such as RTDP or Sarsa, are particularly useful here because they can be adapted to focus their sweeps on

[2]More precisely, we mean reachable within the effective horizon of the MDP; states after the horizon have negligible impact

the relevant states, yielding significant speedup. Unfortunately, it is possible in the general case for the set of reachable states to be the set of all states. As an example, consider an MDP in which every state can transition to every other state, forming a clique. In such cases, partial sweep methods such as RTDP and Sarsa lose any benefit from focusing on the query state, and the difficulty of the narrower problem is the same as that of the general MDP – polynomial in the size of the state space. Fortunately, sampling planners have been developed [64, 68, 139] which can maintain the benefit of the narrower focus. These methods can provide tightly bounded estimates of the optimal action, and have complexity independent of $|S|$. This last point is particularly important in this dissertation because the expanding approach we take to achieve approximate solutions relies on the use of these sampling methods. The expanding approach to solving MDPs is essentially a reduction to a series of state-specific MDPs, structured so that later state-specific MDPs can leverage solutions to earlier state-specific MDPs.

Now let us consider a generalization of the MDP, the POMDP. MDPs model the sequential decision problem as a fully observable problem. That is, it assumes all the information needed to predict what will happen under various actions is observed. This information composes the state. Given the state, history is irrelevant and the transition and reward functions are stationary. Not all sequential decision problems can be well modeled as a fully observable problem, for example, the simple task of navigating in the dark. To address these issues, is a generalization of the MDP known as the partially observable Markov decision problem, or POMDP. MDPs and POMDPs are closely related. First, by adding sufficient history into the state, a sequential decision problem can be modeled as an MDP rather than as a POMDP. Second, under the POMDP model, the problem can be reduced to an MDP through belief states. Belief states capture what state(s) the agent thinks it is in and is represented as a distribution over the set of states. Under belief states, the transition and reward functions are stationary and we recover a well defined MDP. Some of the contributions of this dissertation may be applicable to POMDPs either through this reduction or directly. However, such extensions are beyond the scope of this work and we defer such to future work.

Another generalization of the MDP considers the case of non-atomic actions. This is known as the semi-Markov decision problem or SMDP. SMDPs are often used in works involving hierarchical decomposition because solutions to subtasks form temporally extended actions. Some of the work in this dissertation on leveraging human input involves hierarchical decomposition, and at times I will borrow and make use of the SMDP formalism. However, the focus of our work lies in the MDP and should be viewed in that light. Although some insights may apply to the SMDP, we do not pursue such extensions.

A special case of various MDP models addresses the scenario when transition and reward functions are unknown. Instead, it is assumed that they can only be experienced, *i.e.,* sampled, through interaction with the environment. This closely related problem is the focus of study in reinforcement learning [120]. As a result, many of the methods in RL focus on sample backups or sample based improvements in general. These methods are known as model-free techniques and canonical examples include Sarsa and LSPI. Other methods in RL, known as model-based methods, take a different approach. These methods focus on using experience to build a model of the transition and reward functions, forming a fully defined MDP, and then using existing MDP methods to solve it. The contributions in this dissertation will be applicable to RL through this latter approach. By making improvements to the scalability with which we can solve large MDPs, we will improve our ability to solve large RL problems.

Finally, it is worth mentioning that for some sequential decision problems, it is not always clear how to formulate or author the reward function. Some efforts in the literature have addressed this issue. In particular, work on inverse reinforcement learning [94] eases the authorial burden by automatically inferring an appropriate reward function through demonstrations of the desired behavior. In this dissertation we do not address the authoring of the reward function (or the MDP in general). In the work discussed here, we assume the sequential decision problem has already been formulated as an MDP.

CHAPTER III

APPROXIMATION IN MDPS

Traditional MDP solutions such as value iteration or policy iteration depend on exact, tabular representations. Their complexity is polynomial in $|S|$, the size of the state space. This poses some difficulty for scaling because practical problems tend to have very large state spaces. Recall that the number of states grows exponentially with the number of dimensions of the state space.

To address this difficulty, one might look for hierarchical decompositions or use higher-level representations such as relational representations. These approaches can be highly effective, but only for specific types of MDPs where the special properties they exploit hold. For the general case, approximation methods must be used.

Unfortunately, introducing approximations into MDP techniques is not straightforward. While tabular techniques have been shown to have desirable properties such as convergence and optimality, the use of even simple approximations can introduce errors which jeopardize these properties. This can occur even if the best approximation is found at each step before updating the policy, even over many different notions of "best", from mean-squared-error to residual-gradient [125]. For some cases, approximations have been made safe. Approximators known as averagers, for example, can be used with standard methods like value iteration without losing desirable properties. Policy gradient methods and more recent work in TD learning can guarantee convergence to a local optima for smooth approximators. Finally, approximate LP solutions can guarantee convergence *wrt.* the global optimum for linear approximators, but only when information on the relative importance of states is known *a priori*. Despite many improvements over the years, results are still incomplete. In the general case, the best guarantee for smooth approximators is convergence to a local optima. Surprisingly, this also the best result for the simple linear approximation model.

In this chapter, I tackle the *approximation problem*, the problem of building an effective approximation algorithm with good convergence properties. In particular, I develop a method which, for the class of estimators that can provide confidence intervals, can (1) yield several orders of magnitude speedup over exact methods and (2) guarantee convergence to an optimal or near-optimal policy. This class of estimators includes linear regression and local linear regression, meaning we will be able to approximately but accurately solve any MDP with a smooth value function. Our approach is based on the idea of avoiding *bootstrapping*, the practice of estimating values or actions on the basis of other estimates, which themselves are most likely incomplete and in need of correction.

In the following sections, we begin by gaining a deeper understanding into the difficulties of using approximation, how bootstrapping lies at the root of the problem, and our basic approach which attempts to avoid it. Unfortunately, our approach will introduce certain difficulties, so in the remaining sections we consider two practical methods for addressing such. The first, expansion using worst-case estimates, makes certain compromises which produce fast algorithms but is more limited in its applicability. The second, expansion with sampling planners, uses a more expensive approach but is general and will allow us to prove the bounds we desire.

3.1 Difficulties of Approximation

It is well established that using approximate representations in solving MDPs can introduce instability. Early results showed direct application of approximation to basic algorithms such as policy or value iteration often led to divergence. To see why, let us use policy iteration (PI) as an example. PI creates a sequence of policies starting with some initial policy. Each new policy is generated by one-step lookahead on the value function of the previous policy, creating a process converging to the optimal policy.

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \ldots \pi^* \rightarrow V^*$$

PI is guaranteed to converge because every policy produced is guaranteed to be better than the last unless the last is already optimal [120, 105]. When approximation is used, the story becomes different. When PI computes the value function in some iteration, the

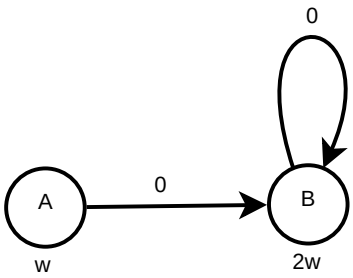

Figure 5: A simple MDP demonstrating divergence for VI. There are two states, A and B. A linear model $V(s; w) = ws$ is assumed where w is the slope parameter. Under each state is its value estimate *wrt.* the weight. There is only a single action and all rewards are zero. Discount is set to 0.9

resulting value function is approximated, incurring some error. When PI uses the (approximated) value function to generate the next policy, that policy is also approximated, incurring additional error. These errors may overwhelm any improvement made, resulting in a subsequent policy that is possibly worse than the original. As a result, convergence is not guaranteed and sometimes, divergence is observed.

One way to avoid policy approximation error is to only represent the value function, *i.e.,* $V^0 \to V^1 \to \ldots \to V^*$. In such value-based methods, the policy is implicitly defined as the greedy policy *wrt.* the current value function. A prime example is value iteration (VI). While this technique removes one source of error, value approximation error remains and in the general case, it may also overwhelm any improvement resulting in divergence.

As a concrete example, we turn to a simple MDP first introduced by Tsitsiklis and Van Roy in [135]. In this problem (see Figure 5), there are two states, A and B, and only a single action. All rewards are zero, and the discount is set to 0.9. In this MDP, the true V^* of both states is zero. For this example, let us use a simple approximation scheme: least squares approximation with a simple linear model of the form $f(s) = ws$, where w is the scalar weight parameter of the model. Clearly, V^* can be captured with our linear model, simply use $w = 0$. If the weight parameter is initialized to this optimal parameter, our approximate VI algorithm is stable. It will remain at the fixed point and return it as the solution. However, if the weight parameter is initialized to any other number, it will diverge to infinity or negative infinity. To understand why, let us see how the value function evolves. Recall the value iteration update equation: $V^{k+1}(s) = \max_a(R(s, a) + \gamma\mathbb{E}[V^k(T(s, a))])$. In

our example, the update equation for the two states simplifies to:

$$V^{k+1}(A) = \gamma V^k(B) \tag{11}$$

$$V^{k+1}(B) = \gamma V^k(B) \tag{12}$$

The least squares estimator for V^{k+1} is then:

$$w^{k+1} = \underset{w}{\operatorname{argmin}}[(w - V^{k+1}(A))^2 + (2w - V^{k+1}(B))^2] \tag{13}$$

$$= \underset{w}{\operatorname{argmin}}[(w - \gamma 2w^k)^2 + (2w - \gamma 2w^k)^2] \tag{14}$$

where our discount factor, $\gamma = 0.9$. Solving this equation we obtain:

$$w^{k+1} = 1.08w^k$$

Thus we see that for any $w^0 \neq 0$, the sequence of value functions produced will diverge.

Efforts have been made to better understand the effect of approximation and to bound its error. Munos in [87] introduces the concept of the *inherent Bellman error* for characterizing the approximation power of a function class *wrt.* an MDP. For a function class $\mathcal{F}$ and MDP with Bellman operator P, the inherent Bellman error for the maxnorm is defined as:

$$d(P\mathcal{F}, \mathcal{F}) = \max_{g \in \mathcal{F}} \min_{f \in \mathcal{F}} ||f - Pg||_\infty$$

It characterizes the largest approximation error that could be incurred between Pg, the one-step improved value function from a function in $\mathcal{F}$, and f, the best approximation in $\mathcal{F}$ to Pg. The inherent Bellman error reflects how well the function space $\mathcal{F}$ is "aligned" to the dynamics of the MDP. This worst-case bound provides an upper bound on the error from one application of the Bellman operator, and allows us to construct a bound on the sequence of value functions as a whole:

$$\frac{2\gamma}{(1-\gamma)^2} d(P\mathcal{F}, \mathcal{F})$$

Applied to our concrete example, we see that the approximation error for one-step

improvement *wrt.* a particular value function $\begin{pmatrix} g \\ 2g \end{pmatrix} \in \mathcal{F}$ is:

$$d_g = \min_{f \in \mathcal{F}} ||f - Pg||_\infty \tag{15}$$

$$= \min_{f \in \mathbb{R}} [\max(|f - \gamma 2g|, |2f - \gamma 2g|)] \tag{16}$$

$$= 1.2g \tag{17}$$

Plugging this in, we see that the inherent Bellman error, $d(P\mathcal{F}, \mathcal{F}) = \max_{g \in \mathbb{R}} d_g = \max_{g \in \mathbb{R}} 1.2g$, is unbounded, matching the observed divergence.

Other analysis [49, 135, 50] focus on understanding convergence. They note that the sequence $V^0, V^1 = \Pi P V^0, V^2 = (\Pi P)^2 V^0, \ldots, V^k = (\Pi P)^k V^0$ must converge whenever ΠP is a contraction mapping. Here Π is the projection operator that maps value functions to the function class of the approximator. Since P is known to be a contraction mapping, as long as Π is non-expansive in the maxnorm, the sequence must converge. A special type of approximators, named "averagers", have this property. They guarantee non-expansion in the maxnorm for all possible value functions. Averagers include approximation schemes such as local weighted averaging and linear interpolation, but not linear regression, local linear regression, nor neural networks. Averagers are distinguished in that they compute the value of a given state as a linear combination of the known values of nearby states. Gordon shows in his dissertation [50] that approximation with averagers guarantees convergence. Averagers essentially build an embedded MDP from its samples and use the fixed point solution of the embedded MDP as its approximation to the actual solution. While bounds on the distance between the embedded fixed point and the optimal solution depend on the exact approximator used and the number of samples available, it is easy to see that in the limit of infinite samples, the embedded MDP will approach the actual MDP and yield the optimal solution. In practice, we note that the embedded MDP must be much smaller than the actual MDP or the complexity of the sampled problem will rival that of the original.

What these analyses make clear is that intermediate value functions cause a great deal of difficulty. This difficulty is the reason that the inherent Bellman error bound is defined *wrt.*

the worst-case approximation error of a function class. It is also the reason that averagers must be non-expansive over the maxnorm for all possible value functions. Because the sequence of intermediate value functions is unpredictable and may very well produce value functions that the approximator is poor at approximating, analysis must cover the worst possible cases. It also means that tailoring the approximator based on domain knowledge of V^* is not sufficient to guarantee convergence.

The source of intermediate value functions and our ultimate culprit is bootstrapping. Recall that bootstrapping is the practice of estimating values or actions on the basis of other estimates, which are most likely incomplete and in need of correction themselves. Bootstrapping methods start from some arbitrary initial policy or value function and rely upon iterative improvement to compute a solution. Due in part to the arbitrary initialization, and in part to the random sampling used to perform improvement, the value functions encountered along the iterative sequence may be arbitrary. Bootstrapping fundamentally requires accurate approximations of not just the final function, but *all possible* intermediate functions. This often leads to erratic behavior in practice. At times, one run will converge to a good policy while another run will fail completely. Sometimes, adding an additional feature to the feature set can trigger complete failure. This behavior makes bootstrapping methods difficult to use.

Bootstrapping also introduces other difficulties. For one, it causes certain inefficiencies. The nature of bootstrapping means that it is computing estimates based on other estimates. If those other estimates are revised, the original computation is wasted and must be recomputed. Ideally, we would like to compute the value of a state only once. For another, bootstrapping has poor anytime behavior. Consider the simple maze example shown in Figure 6 where the goal lies down a narrow hallway. State values in the room cannot be computed accurately until information about the goal propagates down the hallway to the room. As a result, for all but the last few iterations, the vast majority of state value estimates are wrong. Further, it is impossible to tell from the value function, which state estimates are accurate and which are not.

Figure 6: A simple maze MDP with deterministic actions and uniform negative reward of -1. The star at the far right represents a zero valued terminal (*i.e.,* absorbing) state.

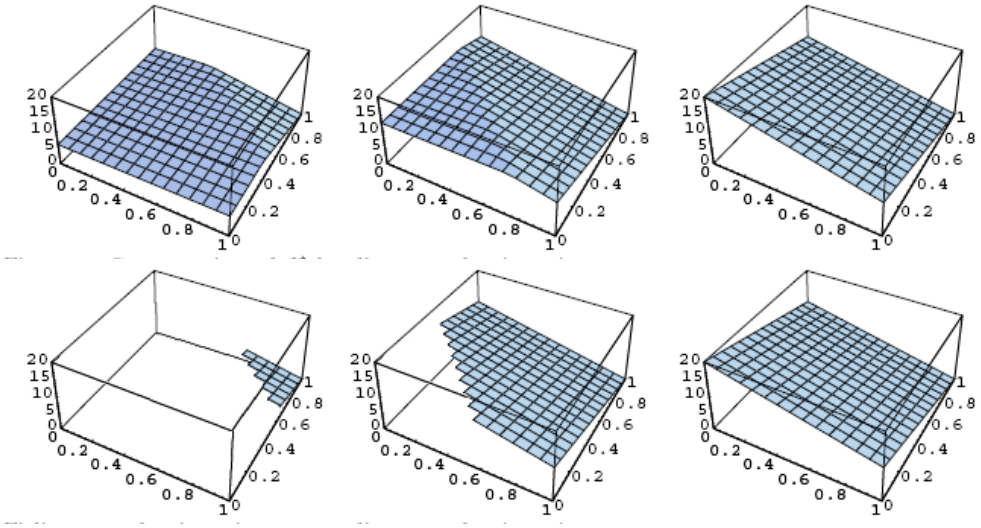

Figure 7: Iterative (top) vs Expansive (bottom) approach to solving MDPs. Illustrations based on [19]

3.2 The Expansion Approach

Our insight is to avoid bootstrapping — to only estimate V^* so that we avoid recomputations and intermediate value functions. To do so in a feasible fashion, our approach is to compute V^* for just a small portion of the state space, and to then expand this region by leveraging previously computed solutions. We call this approach "expanding" as opposed to "iterating" because it works by expanding an accurate approximation of limited size, as opposed to refining an inaccurate approximation over the entire state space.

Figure 7 shows how the expansion approach differs from traditional, iterative methods. Notice how the expansion approach only ever models the optimal value function, albeit incompletely at first. We summarize the benefits of the expansion approach as follows:

- It eliminates approximation of intermediate functions as an error source.

- It offers computational savings from not computing intermediate value functions.
- It offers better anytime behavior: the expansion approach always has an accurate, though perhaps partial, value function.

Eliminating intermediate functions is the key to obtaining good convergence behavior. Intuitively, without interference from intermediate functions, as long as we can represent V^*, we should be able to converge to it. It is easy to see how the expansion approach could achieve this. First, we can clearly compute and represent V^* for terminal, or absorbing states. This provides us with an initial V^* that is accurate, though very partial. Then, as long as sufficient samples are obtained to accurately represent V^* in every expansion, we will maintain an ever growing V^*. Eventually, expansions will cover the entire state space and we will have converged[1] to a complete V^*. This property, basic as it is, is one which bootstrapping approaches cannot offer. In Section 3.4, we will present a specific expansion algorithm, EVA, and prove exactly this guarantee. Further, we will derive theory on how error propagates in EVA to obtain bounds for when the approximator cannot accurately represent V^*.

The expansion approach requires two elements: (1) a set of initial states whose value can be directly computed, and (2) a method of leveraging solved states to expand the solved region. We use the set of absorbing states as the initial set. Expanding the set of solved states poses a greater challenge. The solution to an unknown state, a state outside the solved region, may depend upon other unknown states. In the general case, it is possible for all unknown states to be interconnected such that solving any one state would require the solving of all states, creating a deadlock situation. In the next two sections we will consider two different strategies for addressing this deadlock situation.

[1]The use of the term "converge" can be argued as improper because we are not referring to a sequence of fully defined value functions becoming arbitrarily close to some fixed V, but rather a sequence of value functions, always equal to V where defined, becoming more and more completely defined. Nevertheless, at the risk of abusing notation, we will use the term "converge" because conceptually we are approaching (and eventually reaching) a fixed V.

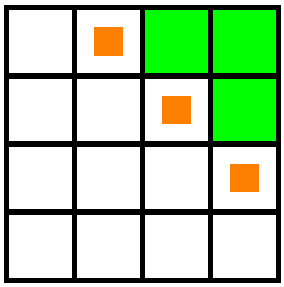

Figure 8: A simple maze example illustrating the pessimistic expansion strategy. A single terminal state is located in the top right corner. Reaching it yields a reward of 1; all other rewards are 0. The green squares represent the set of "known" or solved states. White squares are unknown states whose value is assumed to be $V_{min} = 0$. The states with red centers are the ones that can be solved and added to the known region. These states are solved using a one-step backup.

3.3 Expansion Using Worst-Case Estimates

In this section, we explore a strategy for resolving the deadlock situation which uses worst-case estimates for unknown states and single-step expansions. This strategy enables us to inexpensively estimate the value of arbitrary unknown states; only a single backup is needed to estimate a value. Figure 8 illustrates this strategy in a simple maze example. We call this strategy the pessimistic expansion strategy, or *P-expansion* for short.

P-expansion relies on the assumption that using worst-case estimates for unknown states will not affect the computation of V^*. This is equivalent to assuming that an unknown state s which is one step away from the solved region, has an optimal value $V^*(s) = R(s, \pi^*(s)) + \gamma \mathbb{E}[V^*(T(s, \pi^*(s)))]$ which can be expressed solely in terms of the reward function and the values of states in the solved region. In other words, it assumes that the set of possible next states for s, under an optimal action a, is completely contained within the known region.

Unfortunately, this assumption does not always hold. Consider our maze example in Figure 8. The optimal action for the top red-centered state — let us call this state s_a — is to go `Right`. But suppose this action is stochastic, and has some probability of transitioning to the state below. In such a case, the estimate we compute for s_a would be inaccurate, the computation pessimistically assumes the value of the state below is zero, but its true value is higher. As a result, the computed estimate would be pessimistic and need later revision.

Thus we see that this strategy effectively employs a limited form of bootstrapping. It brokers a compromise between the expansion and bootstrapping (*i.e.,* iterative) approaches.

To better understand the trade-offs of this composite strategy, we first examine its behavior in the exact case, and then consider its application to the approximate case. We will see that P-expansion has certain weakness due to the limited bootstrapping that it relies on, and can only be fruitfully applied to acyclic domains under approximation.

3.3.1 Understanding the trade-offs — the Exact Case

Figure 9 shows P-expansion as applied to value iteration. We call this algorithm, Reverse Value Iteration (RVI) because it can be considered a form of asynchronous VI and has the same theoretical properties, but works by backwards expansions from the terminal state. In keeping with value iteration, only states whose value change by more than some precision, ϵ, are expanded. Note that rather than maintaining a set of all solved states, RVI maintains just the fringe set. This modification is made so RVI can revisit and update previously expanded states as they may be bootstrap estimates in need of correction. Specifically, keeping just the fringe forces expansion in two directions: outward, towards new states, and inward, towards states previously expanded. If previously expanded states are accurate, inward expansion halts and we effectively obtain behavior equivalent to that of keeping the full set of expanded states. However, if previously expanded states yield corrections, then the inward expansion will continue, in case other states also need updating.

The trade-offs made by P-expansion are best illustrated under different cases. These cases are based on the reasonableness of P-expansion's assumption, which we describe along two axes.

- **Optimality:** Whether the estimated best action, assuming worst-case values for unknown states, is an optimal action.
- **Cyclic:** Whether there exists cycles within the solved states, under the greedy policy.

In the next few subsections, we will illustrate each case using RVI's behavior as a guide.

Require: $M = (S, A, T, R, \gamma)$, ϵ
// M - the MDP to solve
// ϵ - the precision parameter

Create initial value function $V : V(s) = V_{min}$, for all $s \in S$
Create the initial fringe from terminal states $F = \{s \in S : \texttt{absorbp}(s)\}$
Solve the initial states $V(s) = \frac{\max_a R(s,a)}{1-\gamma}$, for all $s \in F$
while $F \neq \emptyset$ **do**
  $allparents = \bigcup_{s \in F} \texttt{parents}(s)$
  $estimates = \{(s, \hat{V}(s)) : s \in allparents, \hat{V}(s) = \max_a(R(s,a) + \gamma \mathbb{E}[V(T(s,a))])\}$
  $F = \{s \in S : (s, \hat{V}(s)) \in estimates, |\hat{V}(s) - V(s)| > \epsilon\}$
  Set $V(s) = \hat{V}(s)$, for all $(s, \hat{V}(s)) \in estimates$
end while
return V

Figure 9: Reverse Value Iteration [150, 60]. `absorbp` is an indicator function returning whether its argument is an absorbing state. `parents` is a function returning the set of states which can transition to its argument; it provides an inverse model. An efficient implementation of `parents` which builds a constant-time lookup table is given in the appendices, see Figure 10.

Require: $M = (S, A, T, R, \gamma)$, ϵ

Initialize inverse model: $parents = \{s \rightarrow \emptyset : s \in S\}$
for all $s \in S$ **do**
  for all $s' \in \{s' \in S | T(s,a)(s') > 0, \forall a \in A\}$ **do**
    $parents(s') = parents(s') \bigcup s$
  end for
end for

Figure 10: Efficient implementation of a lookup table inverse model

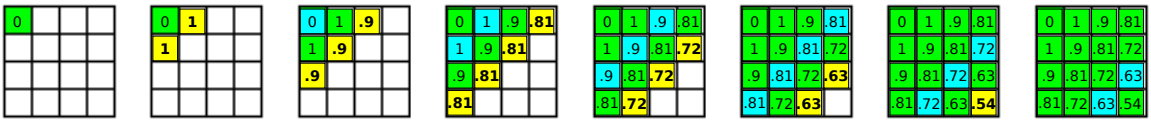

Figure 11: Progress of RVI in a simple grid-world. A single terminal state is located in the top left corner. Reaching it yields a reward of 1; all other rewards are 0. The discount factor is set to 0.9. V^0 shows the initial value function holding just the value of the terminal state. Empty cells represent states without an estimate, green cells represent states with previously computed values, and yellow/blue cells represent states whose values have just been (re)computed — the states being expanded. Yellow cells track the outward expansion and blue ones track the inward expansion. Computations which yield changes are highlighted in bold.

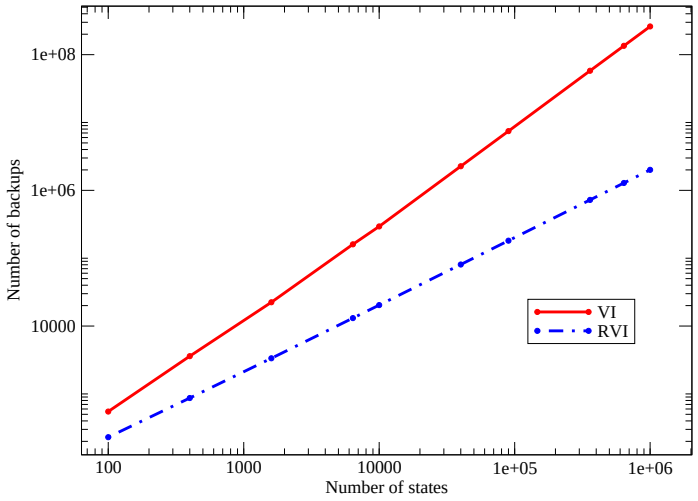

Figure 12: Comparing the expansion based RVI with VI on grid-worlds of various sizes

3.3.1.1 Optimal and Acyclic

Let us first consider the expansion strategy when all assumption hold. In this case, expansion estimates are both optimal and acyclic. Figure 11 shows the progression of RVI in such a domain. The blue (inward) expansion states can be ignored; they never propagate beyond the first set of parent states. The yellow (outward) expansion states illustrate the progression of RVI. The expansion strategy works flawlessly. The (outward) expansion fringe travels in concentric rings across the state space, solving states and adding them to the known region. The solved region grows as an enlarging circle, until all states have been covered.

Note that all value estimates are final — they are never revised. In other words, no bootstrapping estimates are used. All the benefits of the expanding approach are evident in this case. There are no intermediate value functions so there is no possibility of errors from approximating them and no computation is wasted on computing them. We also observe excellent anytime behavior — at any point in time the value function, where defined, is always accurate.

Empirical experiments confirm the benefits of P-expansion. Figure 12 shows the performance of RVI compared to that of VI. Note how RVI scales linearly with the number of states. This is made possible by the avoidance of bootstrapping estimates. Each state

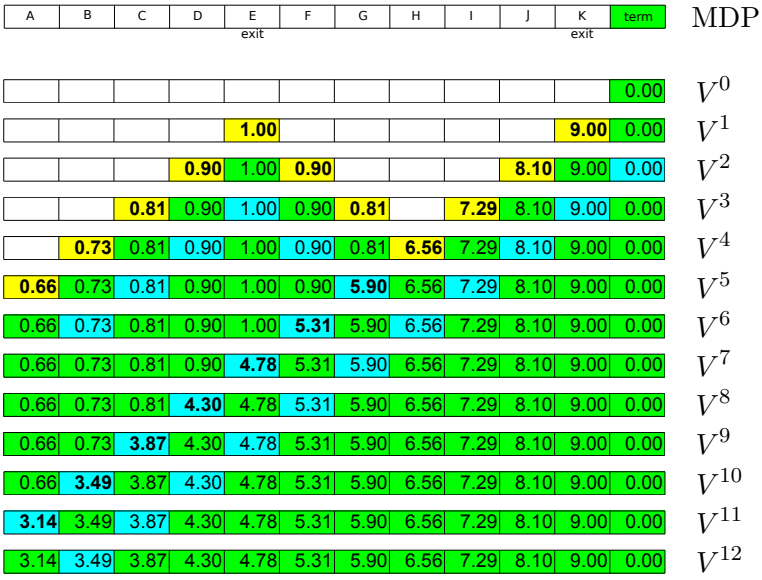

Figure 13: Progress of RVI in a hallway grid-world with two exits, *i.e.,* two states which can go to the terminal state. The middle exit yields a reward of 1, the far right exit yields a reward of 9. All other rewards are 0. The discount factor is set to 0.9. V^0 shows the initial value function holding just the value of the terminal state. Empty cells represent states without an estimate, green cells represent states with previously computed values, and yellow/blue cells represent states whose values have just been computed — the states being expanded. Yellow cells track the outward expansion and blue ones track the inward expansion. Computations which yield changes are highlighted in bold.

needs just two backups — the first to compute the value, and a second to confirm that it is the correct and final value.

3.3.1.2 Suboptimal and Acyclic

Now let us consider the case when use of worst-case estimates leads to suboptimality. Figure 13 shows the progression of RVI in a simple hallway grid-world. We see that in the very first expansion round, the value of state E is computed incorrectly. The optimal action is to go right, to state F, but since the value of state F is estimated as $V_{min} = 0$, the immediate exit action appears to be the best. Over the next few expansion rounds this incorrect estimate is propagated to states C through G. It is not until accurate estimates catch up in round 5, that these mistakes are corrected. These corrections are possible because RVI keeps only the fringe and expands inward into previously expanded states to catch just such mistakes.

From this example, we see that while P-expansion does make mistakes when worst-case

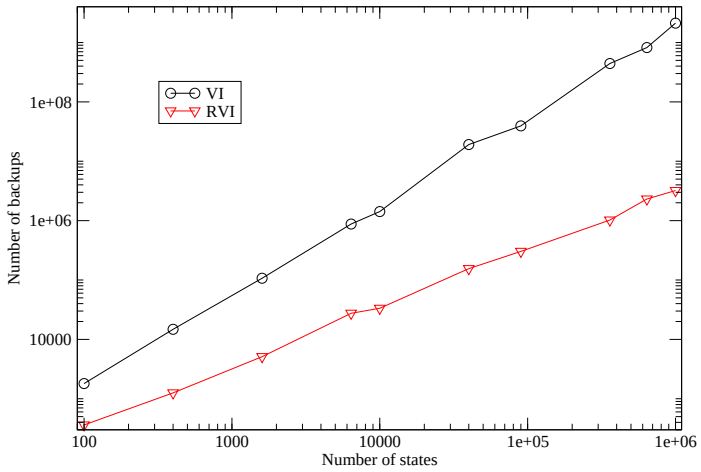

Figure 14: Comparing the expansion based RVI with VI on grid-worlds with randomly chosen exit states.

estimates are incorrect, by checking solved states, we can detect and correct these mistakes. Overall, the soundness of the expansion strategy is maintained, but there is some additional cost. Specifically, some of the benefits of the expansion approach are (partially) lost:

- Some intermediate value functions are computed. In our example, incorrect value functions are computed in rounds 1-4. Note however, that even the incorrect value functions are the V^{π} for some (suboptimal) policy π. This is in contrast to bootstrapping methods where one may encounter, and need to approximate, value functions which do not correspond to any policy.

- Due to the computation of some intermediate value functions, some waste will occur. However, the added cost is at most a constant multiplier. In every expansion round, at least one state must be computed correctly. Further, when a mistake is detected, propagation from states with lower values can be temporarily halted. This will prevent mistakes from propagating while correct values catch up, reducing some waste.

- Anytime behavior is degraded. Expanded states are no longer guaranteed to be V^*. However, the estimate can still be useful: (1) it is the true value of some policy, and (2) it provides a lower bound *wrt.* the optimal value function.

In empirical experiments, RVI shows similar speedups as in the previous case. Figure 14 compares RVI and VI in various sized grid-worlds with random exit states. These exit states

transition to the terminal state with a randomly chosen reward and are placed randomly. This causes RVI to perform suboptimal computations like those in our hallway example in Figure 13. Results show that RVI maintains a linear scaling *wrt.* the number of states. By the time VI requires millions of backups, RVI is achieving speedups of around two orders of magnitude. These results imply that although some limited bootstrapping is necessary in the suboptimal case, the vast majority of bootstrap estimates are avoided. In other words, the majority of the advantages of the expansion approach are maintained.

3.3.1.3 Cyclic Cases

Cyclic cases are the worst cases for P-expansion. They can only occur in stochastic environments. Figure 15 shows the progression of RVI in a similar hallway grid-world which both contains cycles and encounters suboptimality.

When expansion reaches the cannon state in round 6, it results in inward backups to states G and H, in rounds 7 and 8. The values of these two states are initially based on going to the right, but the optimal action is to go left to the cannon state. As a result, their values undergo corrections when they are updated. Because the value of the cannon state also depends on G and H, when they are updated, the cannon state's value must also be recomputed as well. This cycle between states F, G, and H requires multiple recomputations before updates fall below the precision threshold (set to 0.01 in our example). Within the cycle, RVI's behavior devolves to that of VI. In general, when cycles are encountered, the cycle region becomes a miniature subproblem in which expansion cannot be used and the standard bootstrapping approach must be used instead. To make matters worse, while the cycle region is being recomputed, intermediate changes are continually propagated by the expansions. These expansions are a waste as they propagate incorrect intermediate values from the cycle region. Ideally, we would like to wait until the cycle region converges before propagating the values within, but because cycles can occur over an arbitrary number of rounds and an arbitrary number of states, detecting them is expensive and unlikely to yield a net benefit.

In summary, all benefits of the expansion approach are lost within cycles. Outside

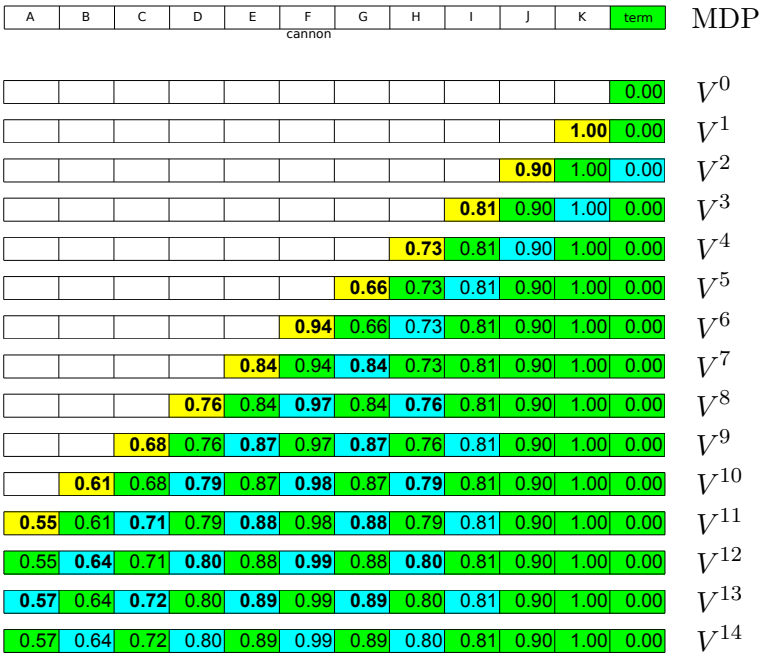

Figure 15: Progress of RVI in a simple grid-world. A single terminal state is located on the far right. The middle state F has a `cannon` action in addition to the standard `left` and `right` actions. Taking the `cannon` transitions to one of the non-terminal states to the right, $G, H, \ldots, K$, with equal probability. Reaching the terminal state (only possible by going `right` from state K), yields a reward of 1; all other rewards are 0. The discount factor is set to 0.9. V^0 shows the initial value function holding just the value of the terminal state. Empty cells represent states without an estimate, green cells represent states with previously computed values, and yellow/blue cells represent states whose values have just been (re)computed — the states being expanded. Yellow cells track the outward expansion and blue ones track the inward expansion. Computations which yield changes are highlighted in bold.

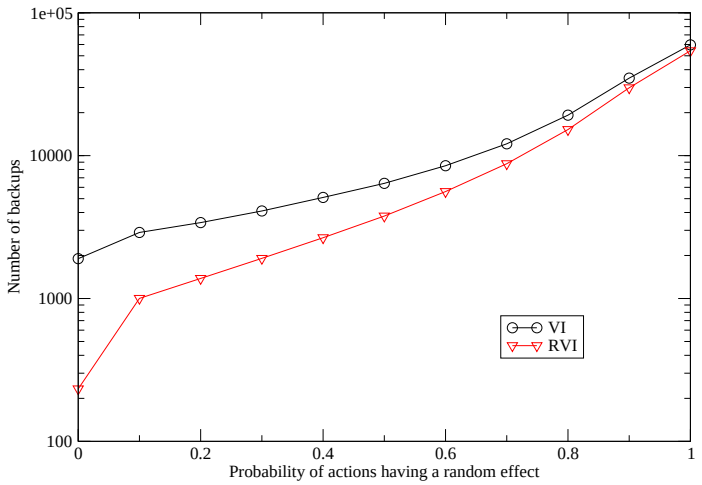

Figure 16: Comparing the expansion based RVI with VI in a simple 10x10 grid-world with different levels of randomness.

cycles, some benefits are also lost because intermediate values generated by cycles pollute expansions to acyclic states. Overall, the larger the cycle (*i.e.,* the more states involved), the more benefit is lost. In the worst case, the expansion strategy may offer no benefits.

Empirical results confirm the effect of cycles. Figure 16 shows the performance of RVI and VI over different levels of randomness on a simple 10x10 grid-world. When an action is taken, it either transitions deterministically in the direction specified by the action (*e.g.,* `left`, `right`, `up`, or `down`), or it transitions (uniformly) in a random direction. As we increase the probability of the random effect, the benefits of the expansion approach, RVI, disappear. This should come as no surprise; increasing the probability of the random effect increases the effective size of the cycle, and we know from analysis that larger cycles are more difficult for RVI. When actions are completely random, the performance of RVI and VI become identical.

3.3.1.4 Comparisons on Realistic Domains — A Mixed Setting

Empirical comparisons thus far have been based on synthetic grid-world domains. These have been useful for illustrating the differing behavior of our expansion strategy under different cases, and for understanding the worst case and best case possibilities. However, they do not provide much intuition for the more common situations. To address the more common scenario, we present empirical results on several common RL benchmark domains:

Mountain Car (MCAR), Single Armed Pendulum (SAP), and Double Armed Pendulum (DAP). These domains model real, albeit simplified, physical processes. While these underlying (continuous) processes are deterministic and thus acyclic in nature, discretization introduces stochasticities and cycles. As a result, these domains can be considered a mix of cyclic and acyclic cases. Details of the domains can be found in Appendix A.

In the following results, in addition to comparing against traditional VI, we also compare against other state-of-the-art value iteration based methods: BVI [30], FVI [30], and two variations of GPS [142, 144], H1 and H2V[2]. Backwards Value Iteration (BVI) and Forwards Value Iteration (FVI), are heuristic methods that use the state transition model and a policy to modify update ordering. This can have the effect of avoiding certain bootstrap computations. The policy used is assumed to be given *a priori*. If none is available, the greedy policy of the estimated value function is used. In this case, the heuristic focuses updates on states with high values. The two General Prioritized Solver (GPS) methods, H1 and H2V, are also heuristic methods which reorder or prioritize backups. However, GPS first partitions the state space and prioritization is performed on top of partitions. Partitioning helps to minimizes prioritization overhead and can help limit bootstrapping backups to within partitions. The H1 algorithm prioritizes those partitions with high Bellman error. The H2V algorithm prioritizes those partitions with high Bellman error and high values. H2V also uses a special static ordering of backups within partitions computed based on a form of topological sort.

Table 1 shows the results. First, notice that RVI performs better than VI, and particularly excels under fine discretizations (*i.e.,* more states). This is in line with our cyclic case analysis. Finer discretizations decrease the amount of stochasticity in the domain, which leads to fewer and smaller cycles, resulting in higher performance.

Second, notice the consistent behavior of RVI. It is *always* faster than VI — usually about an order of magnitude faster. The other methods do not behave as consistently. H2V performs very well on all MCAR and SAP variations, but is actually twice as slow as VI on

[2]H1 refers to the GPS algorithm with the H1 heuristic. H2V refers to the GPS algorithm with the H2 heuristic plus voting.

Table 1: Comparison of the performance of VI, RVI, BVI, FVI, H1, and H2V, in thousands of backups required. The best result for each domain is highlighted in bold. Results for H1 and H2V are taken from [142, 144] and are only precise to the millions. MCAR, SAP and DAP refer to the Mountain Car, Single Armed Pendulum, and Double armed Pendulum domains, respectively. Details on these domains can be found in Appendix A.

Domain : # states	*VI*	*RVI*	*BVI*	*FVI*	*H1*	*H2V*
MCAR : 40k	9,575	**1,415**	3,955	4,715	2,000	**1,000**
MCAR : 90k	31,477	3,467	11,362	14,393	6,000	**1,000**
SAP : 40k	11,765	**1,278**	1,760	7,959	4,000	**1,000**
SAP : 90k	34,312	3,227	13,410	23,129	9,000	**1,000**
SAP : 160k	73,100	5,930	11,000	54,636	15,000	**2,000**
DAP : 160k	5,980	**2,733**	5,680	13,029	8,000	11,000
DAP : 810k	33,918	**3,213**	27,945	67,681	16,000	23,000

DAP-160. FVI and H1 have similar cases where they perform worse than VI. This behavior results from the fact that these methods rely on backup ordering heuristics. When the heuristics are a poor match for the domain, they can starve important states of updates. By contrast, RVI uses a systematic expansion strategy which precludes state starvation. As a result, RVI's worst-case complexity can be no worse than that of VI's.

Finally, we note that RVI performs competitively on all domains. While it is not always the fastest, it is not far behind — and on DAP-810, it offers significantly better performance than the rest. In sum, these results suggest that while the expansion strategy works best in acyclic cases, it can still be beneficial in mixed settings typical of many realistic domains.

3.3.2 Application to the Approximate Case

Analysis under the exact case provides us with an understanding of when P-expansion works best and why. The strong performance of RVI is also suggestive. Now we will tackle our main objective: applying the expansion strategy to build an effective approximation algorithm with good convergence properties. We consider P-expansion in the approximate setting using the same breakdown of cases as before.

- **Acyclic and Optimal:** This case is ideal for approximation. No bootstrapping occurs and all estimates are final. There are no intermediate value functions. All the benefits of the expansion approach are in evidence and we should be able to provide guarantees such as convergence to V^*, assuming the approximator can represent it.

- **Acyclic and Suboptimal:** In this case, one may need to approximate V^π for some suboptimal policy π. Although this is a much more limited set of possible intermediate value functions than those that may be encountered under traditional bootstrapping approaches, approximating any intermediate functions can jeopardize convergence to V^*. Even a single intermediate value function could potentially induce an arbitrarily poor approximation that precludes further accurate expansions, preventing convergence to (a complete) V^*. In other words, convergence to V^* can only be guaranteed if the approximator can represent V^* *and* V^π for every encountered suboptimal policy, π.

  This conclusion is interesting but difficult to apply; how does one know what suboptimal policies will be encountered? Here, the structure of the expansion approach provides some help. Specifically, if a difficult-to-approximate value function is encountered, the expansion process will prematurely terminate. Assuming the approximator can represent V^*, this means an expansion run will either (1) stop early and return a partial V^π for some policy π, or (2) finish and return a complete value function which will be optimal with high probability. This behavior is not ideal, but is still preferable to bootstrapping methods, which may not converge to any V^π, and cannot guarantee optimality even when they do converge.

- **Cyclic cases:** Cyclic cases present a worst-case scenario for the expansion approach. As we saw in exact analysis, cycles effectively create miniature subproblems in which bootstrapping must be used to solve. Since cycles can cover the whole of the state space in the general case, the expansion strategy offers little help and no improved guarantees whatsoever.

Our analysis shows that when approximators are in use, the expansion strategy is only viable in acyclic domains. Cycles force reversion to traditional bootstrapping behavior. Thus we will limit our discussion of P-expansion with approximation to the acyclic case.

Under this narrower scope, applying P-expansion with approximation is relatively straightforward. In fact, we can derive it directly from the exact case algorithm, RVI. There are

just two issues that need our attention: the absence of an inverse model, and the need for a test to see if a state has already been solved (*i.e.,* within the expanded region).

In RVI, an inverse model is used to generate *expansion states* from the solved region[3]. Expansion states refer to states which are one step removed from the solved region, *i.e.,* those highlighted in yellow in Figure 11. RVI identifies expansion states by following the parents of solved states. This mapping between states and their parents, *aka.* the inverse model, is easily computed in the exact case because the number of states is limited. In the general case, however, it is not so easy to compute and will require approximation. Rather than requiring an additional approximator and dealing with possible errors in the approximated inverse model, we use rejection sampling instead. We sample states uniformly and, for each state sampled, we test to see if all possible next states, under the estimated best action, are within the solved region. If so, the sampled state is accepted and added to the set of states to expand. Otherwise, it is rejected and another state is sampled. In practice, to reduce the number of random samples that must be taken, we will build a large, fixed set of samples *a priori* and simply iterate through the set to obtain sample expansion states.

The second issue we need to resolve is a way of keeping track of the expanded region. In the exact case, this is trivially solved by simply keeping a list of expanded states. In the general case, this is not an option. The set of states may be too large. To resolve this difficulty, we will rely on rollouts of the value function approximator itself. Given a query state s, we will perform rollouts based on the greedy policy *wrt.* the approximated value function. If rollouts confirm the estimated value of s, *i.e.,* there are no Bellman errors on states along the rollout, then we consider the state to be solved and in the expanded region. We call this the *verification* procedure. Note that the procedure can only guarantee a lower bound on the value. That is, verification guarantees that the estimated value is the true value $V^{\pi}(s)$ for some policy π, but not necessarily for an optimal policy. Rollouts of a specific policy (in this case the greedy policy) can only verify the consistency of states (and

[3]Technically, RVI uses the fringe of the solved region to allow for corrections to expanded states. This will not play a role here as corrections will be performed using a different process. For simplicity, we will ignore the distinction and treat RVI as if it expands from the whole set of solved states.

Require: $M = (S, A, T, R, \gamma)$, Ω, ϵ
  // M - the MDP to solve, it must be acyclic. Assumes non-negative reward.
  // Ω - the set of sample states
  // ϵ - the precision parameter

  $trainset = \{(s, 0) : \texttt{absorbp}(s), s \in S\}$
  repeat
    $\hat{V} = \texttt{mkVF}(M, trainset, \epsilon)$
    for all $s \in \Omega$ **do**
      $estV = \max_a(R(s, a) + \gamma\mathbb{E}[\hat{V}(T(s, a))])$
      if $(estV == \perp)$ **then continue**
      if ($(s, v) \in trainset$ and $|v - estV| > 2\epsilon$) **then** {
        Remove $\{(s, v) \in trainset : v \leq estV\}$ from $trainset$
        $trainset$.add(s, $estV$)
        continue }
      if ($(s, v) \notin trainset$) **then** $trainset$.add(s, $estV$)
    end for
  until $trainset$ stops changing
  return $\hat{V}$

Figure 17: The basic Approximate-RVI or ARVI algorithm. `absorbp` is an indicator function which returns whether its argument is an absorbing state. `mkVF` is a function which uses supervised learning to construct a partial approximate value function (see Figure 18 for details). Note that this means $\hat{V}$ is a *partial* value function, *i.e.,* it yields $\perp$ where undefined.

their values) along that policy. Verifying optimality would require comparisons to other policies and would be too expensive to perform. As an example of verifying suboptimal values, consider state F in round 2 of our RVI example (Figure 13). Despite the incorrect value, it would pass verification because only the greedy policy is considered. Although this property is not ideal, it does not present any additional difficulties — it reduces to the same issue as that of the acyclic and suboptimal case.

Figure 17 shows the basic approximate expansion process of Approximate RVI or ARVI. ARVI corresponds closely to RVI. The while loop in both algorithms compute value estimates for expansion states and adds them into the solved set. In RVI the entire set of expansion states is solved per loop and the results are inserted into the value table. In ARVI the sampled expansion states are solved and added into the training set. This process of sampling and solving states can also be thought of as generating and labeling examples.

One important aspect of ARVI we have yet to discuss is error handling. Due to possible suboptimalities, some expanded states may occasionally need to be recomputed. RVI solves

Require: $M = (S, A, T, R, \gamma)$, *trainset*, ϵ
// M - the MDP to solve, it must be acyclic
// *trainset* - the training set of state-value pairs
// ϵ - the precision parameter

$\tilde{V}$ = train supervised learner on *trainset*
$\hat{V} = \{$**function** $s \rightarrow$
rolloutStates = `rollout`(M, $\tilde{V}$, s)
consistent = iff $|\tilde{V}(s) - P\tilde{V}(s)| \leq \epsilon$, for all $s \in$ *rolloutStates*
if (*consistent*) **then** *rolloutEst* **else** $\perp$ $\}$
return $\hat{V}$

Figure 18: The `mkVF` function. It uses supervised learning to construct a partial approximate value function. In the constructed value function, $\hat{V}$, a query state's (estimated) value is only returned if it can be verified by rollouts. Otherwise, $\perp$ is returned indicating that the value function is not defined at the state. `rollout` performs rollouts based on the greedy policy *wrt.* $\tilde{V}$, and returns the set of states encountered. The variable *consistent* specifies whether the Bellman error for all rollout states are below precision parameter ϵ. P is the Bellman operator.

this issue by forcing exhaustive expansions inward towards solved states. ARVI simply iterates through all sampled solved states and checks whether the Bellman error exceeds the precision parameter. If an outsized error is discovered, it is corrected and inserted into the training set. Further, all states with value lower than the corrected value are removed from the training set. This prevents possibly erroneous values from being propagated while the corrected value catches up.

Although we independently developed ARVI, excepting a few differences it turns out to be a reinvention of two previous algorithms Grow-Support and ROUT, first presented by Boyan in 1995 and 1998 [19, 18]. Specifically, ARVI can be considered a general form of Grow-Support and ROUT. Grow-Support specializes in deterministic domains and is unique in that it simply uses the rollout value estimate instead of performing consistency checks on rollout states. ROUT specializes in stochastic and acyclic domains, and does perform consistency checks. It is unique in that it uses states encountered during rollouts as a source of additional sample states. Finally, Grow-Support and ROUT do not perform error correction. This means they can produce complete, but suboptimal value functions. By contrast, with error correction, when a complete value function is produced, it must be the case that all sampled states have Bellman errors within ϵ. Thus we can guarantee

that for a randomly sampled state s, the Bellman error of s will be within ϵ with high probability. If all states have Bellman error within ϵ then we can bound the value function error: $|\hat{V} - V^*|_\infty \leq \frac{\epsilon}{1-\gamma}$.

Boyan provides a wealth of results for ARVI. For example, he performs several empirical studies on domains ranging from a dice game to production line scheduling and demonstrates optimal or near-optimal results across all domains [18]. Boyan also provides convergence results. Unfortunately, while ARVI is guaranteed to converge, the value function it converges to is not always optimal. If ARVI converges after all samples have been solved and added to the training set, then we can guarantee that randomly sampled states will have error within ϵ with high probability[4]. However, if ARVI converges before then, the value function produced may not be fully defined, and may be suboptimal where defined.

Although convergence results are not ideal, Boyan shows that in practice, these algorithms perform well. The main drawback of this expansion strategy is the limitation of applicability to acyclic domains. This will brings us to our second expansion strategy.

3.4 Expansion Using Sampling Planners

In this section we consider a different strategy for resolving the expansion deadlock situation. Rather than using single backups and pessimistic estimates for unknown states, we will use sampling planners which compute exact estimates of state values without any value assumptions on unknown states. Recall that sampling planners solve the state-specific MDP problem; they directly estimate the value of a given state. Further, they have complexity independent of the size of the state space, so they will not jeopardize the goal of approximation. Figure 19 provides an illustration of how a sampling planner might work. We call this the exact expansion strategy, or E-expansion for short, because we use sampling planners to compute exact estimates rather than pessimistic ones.

Sampling planners are clearly more expensive than singular backups, but they make no assumptions on the values of states, allowing us to completely avoid bootstrapping. This point is particularly attractive because even the slightest use of bootstrapping can

[4]This assumes error correction is being used.

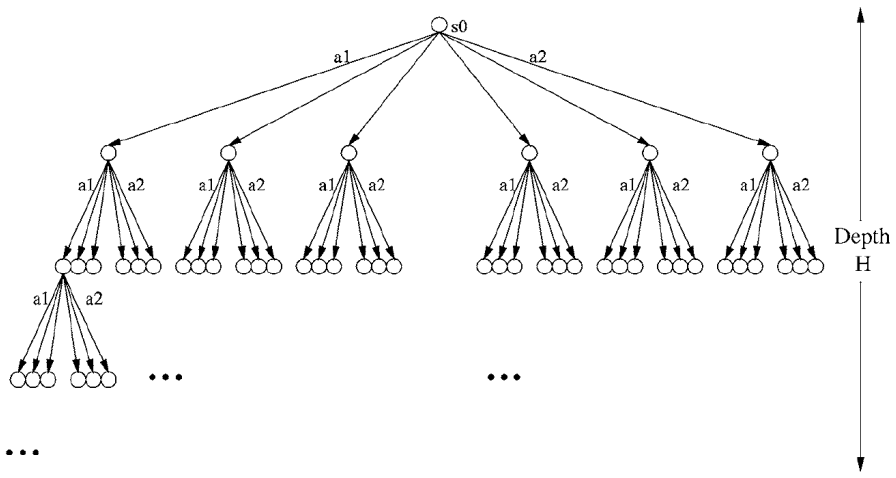

Figure 19: The sparse lookahead tree for a sample planner. Starting from the query state, $s0$, the result of taking each action is sampled multiple times. The resulting next states are then expanded themselves. The tree is generated out to the depth, *aka.* horizon, of the problem. It is then solved by propagating values from the leaves. The final value propagated to the root is the solution value estimate returned by the sample planner. Graphic courtesy of [64].

cause significant difficulties. As we saw in P-expansion, despite limited use, bootstrapping was sufficient to remove all benefits of the expansion approach in cyclic domains, and to weaken optimality guarantees in the "acyclic and suboptimal" case. Using sampling planners essentially puts us in P-expansion's equivalent of the "acyclic and optimal" case.

For rest of our discussions, it will be more convenient to think in terms of cost, G, and cost-to-go, J. Recall that cost is the negation of rewards, and similarly, cost-to-gos are the negation of state utilities. We will also assume, without loss of generality, the existence of a single terminal state and that costs are strictly positive, in the range (0,1], excepting the 0 cost self-loop of the terminal state. This means the cost-to-go of the terminal state is 0, and the cost-to-go of all other states must be greater than 0. We will also assume that costs are between 0 and 1.

In the following sections we will consider E-expansion in detail. We will begin by considering some of the issues that arise from using sampling planners. We will then cover the actual algorithm, EVA. Theoretical analysis will come next. We will prove convergence to J^* when the approximator can represent J^*, and error bounds on J^* when the approximator cannot. Finally, we will present some empirical results and demonstrate that EVA can

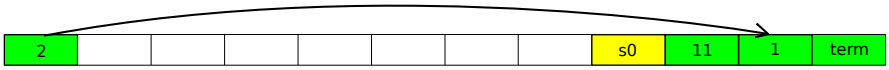

Figure 20: A simple hallway example demonstrating the need to explore the longest policy to guarantee optimality. Actions in non-terminal states have cost 1, except the state two to the left of the terminal state which has action costs of 10. Green cells represent the solved region, showing state values. The yellow cell represents the query state. The optimal policy from $s0$ is to go to the far left. As a result, an optimal planner must consider depth up to the length of the longest possible policy.

provide many orders of magnitude speedup over tabular methods. We also compare against other approximation methods for context.

3.4.1 Issues in Using Sampling Planners

Using sampling planners raises several issues: computational tractability, choosing an expansion size, and verification. We will address each in turn.

3.4.1.1 Computationally Tractability

As Figure 19 shows, sampling planners are exponential in the depth of the problem. A simple way to alleviate this computational cost might be to expand just those states that are a single step away from the solved region. The hope is that this will limit the effective depth of the sampling planner — once it reaches a state within the solved region, it can halt. If we are interested in satisficing solutions (*i.e.,* those that will reach the terminal state, but not necessarily optimally), this is sufficient. However, if we are interested in optimal solutions, starting close to the solved region in and of itself is not enough. Figure 20 illustrates the reason why. In essence, there may be multiple policies which go to the solved region, some very long, and all must be pursued to ensure optimality. As a result, despite starting close to the solved region, the depth of the planner is bounded by the longest possible policy. By policy length, we refer specifically to the expected number of steps required under the policy to reach a target — in this case, the solved region. Note the expectation is well defined because the number of steps is bounded by the horizon of the problem.

To resolve this difficulty, we will perform expansions based on cost-to-go rather than number of steps. In other words, instead of generating a series of expansions that are 1-step,

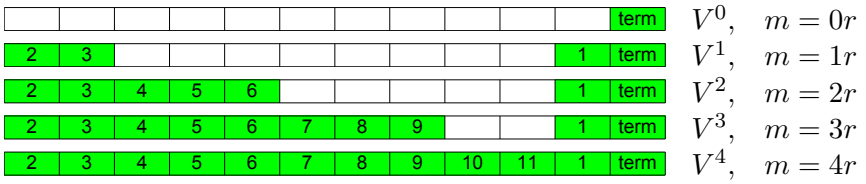

Figure 21: Progression of cost-to-go expansions on the hallway problem in Figure 20. Each row shows the state of the expanded region and its partial value function at the end of an expansion round. m denotes the MERP; radius $r = 3$.

2-steps, ..., k-steps away from the terminal state, we will generate a series of expansions that have cost-go-to within $r, 2r, \ldots, kr$ of the terminal state. Here, $r \in \mathbb{R}$, the radius, is a constant specifying the expansion size, and $k \in \mathbb{Z}$ is some positive integer indicating the number of expansion rounds. Figure 21 shows how expansion *wrt.* cost-to-go progresses. After some expansion round k, the (partial) value function grows to include all states with cost-to-go within kr. We call the cost-to-go boundary, kr, the Maximum Effective Radius of the Predictor, or MERP for short. MERP exactly delineates the expanded region. States with cost-to-gos within MERP are solved and within the region. Those with cost-to-gos beyond MERP are outside the region and considered unknown. In addition to delineating the expanded region, the MERP is useful because it provides a minimum cost-to-go for unknown states. In other words, it can be used as an admissible heuristic for unknown states. This reduces the depth a sampling planner must descend. Figure 22 illustrates the required depth to solve a state in the second expansion round of the hallway problem. Note how all expansion states (denoted by an orange dot) must have cost-to-go in the range $(1r, 2r]$. States with cost-to-go within $1r$ must already have been solved. States with cost-to-go greater than $2r$ are outside the expansion because we expand by r at a time. As a result, in this deterministic example, the required depth for any expansion state is limited by $r/G_{min} = 3$ where $G_{min} = 1$ is the smallest per-step cost. In the general case, the depth limit is more complex due to potential stochasticity in the transitions. The general case depth limit will be covered in Section 3.4.3 when we present the theoretical analysis of EVA.

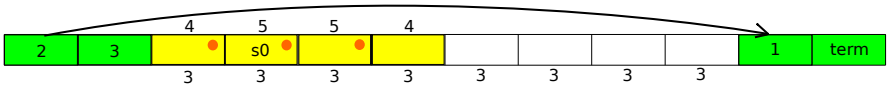

Figure 22: Expansion round 2. The states that must be expanded to solve $s0$ are highlighted in yellow. Green indicates the solved region, grown from previous expansion rounds. Its partial value function has MERP $m = 3$. Orange dots mark expansion states, *i.e.,* the set of states to be expanded, of the current expansion round. Numbers below the states are admissible heuristics (using the MERP) for unknown states. Numbers above the states are final values computed by the sampling planner in the course of solving state $s0$. Unlike before, the depth of the sampling planner is limited by its proximity (*wrt.* cost-to-go) to the solved region.

3.4.1.2 Choosing an Expansion Size

When using singular backups to compute state values, we are limited to one-step expansions. States further than one-step away from the known region cannot be solved because they depend on values of unsolved states. Sampling planners do not have this limitation; they will sample forward to the horizon of the problem if need be to compute the value of a state. This means arbitrarily sized expansions are possible. In theory, we could make the entire state space a single expansion, reducing our approach to a trivial two step process: (1) generate many state-value examples using the sampling planner, and (2) use a supervised learner, *aka.* an estimator, on the examples to learn an approximate value function. In practice, this is not feasible because sampling planners are exponential in the depth of the problem. They can only be tractably used to label or solve states that are close to the solved region. There is a trade-off in choosing an appropriate radius or expansion size. Larger expansions mean fewer total expansions will be needed. They also mean more efficient rejection sampling because each expansion will cover a larger portion of the state space. However, larger expansions are also deeper, which raises the cost of labeling example states. We can also view the radius as imposing a cost-to-go based decomposition on the MDP so that the planner is only ever asked to provide solutions to incremental problems. In this view, the radius controls the granularity of that decomposition. The optimal expansion size is domain-specific and depends upon properties such as the effective branching factor, and the distribution of state values. As a practical matter, we typically choose a radius

that limits planning depth to a few steps deep.

3.4.1.3 Verification

Fundamental to the expansion approach is the use of previously solved states, or rather, their cost-to-go estimates, to speed up computation of new, unsolved states. The problem is determining which cost-to-go estimates are reliable and can be used, and which are not. We call this the verification problem.

One method of verification is to track the expanded region. States within the expanded region have been previously solved or approximated and are thus reliable. When the size of the state space is small, the expanded region can be easily tracked by keeping a list of expanded states. When the state space becomes large, however, tracking the expanded region becomes more difficult. In P-expansion, we used rollouts to track the expanded region. Rollouts, however, can only verify upper bounds, which can create errors in tracking.

In E-expansion, we will use an alternative verification method. We will require that our approximators provide confidence intervals on their estimates, and rely on those intervals to determine reliability. This method is not as general, because it imposes restrictions on our choice of approximators; however, unlike rollouts, it avoids creating errors. As a result, this strategy does not require error handling and, as we will see, provides better error bounds.

3.4.2 EVA Algorithm

The EVA algorithm is shown in Figure 23. Conceptually, the algorithm is quite simple. It begins with a very small, but accurate cost-to-go function containing just the set of absorbing states. This cost-to-go function returns the true cost-to-go, 0, of any absorbing state and $\perp$ otherwise. Returning $\perp$ is the equivalent of indicating "I don't know". This initial value function is then expanded over many rounds until it is complete, *i.e.,* until it can return a cost-to-go estimate for every state in the MDP. Each expansion is composed of three basic steps. First, states are sampled from the larger region to be covered. Second, each of these states are labeled through calls to the sampling planner, *i.e.,* the cost-to-go for each state is computed. Finally, these state-value pairs, or labeled examples, are fed as training data to a supervised learner and the result is used to construct a partial value

Require: $M = (S, A, T, R, \gamma)$, D, L, r, ϵ_p, ϵ_l, δ
  // M - MDP to solve
  // D - sampling planner
  // L - learner (*aka.* estimator)
  // $r \in \mathbb{R} > 0$ - radius controlling the expansion size
  // $\epsilon_p \in \mathbb{R} \geq 0$ - planner precision parameter
  // $\epsilon_l \in \mathbb{R} > 0, \delta \in (0, 1]$ - precision parameters: must be accurate within ϵ_l,
  // with probability at least $1 - \delta$

  $\hat{J}_0 = \{$ **function** $s \rightarrow$ **if**(`absorbp`(s)) **then** 0 **else** $\perp$ $\}$
  $merp_0 = 0$
  for ($i = 1$; $\hat{J}_i$ is incomplete; i++) **do**
    $merp_i = merp_{i-1} + r$
    $C_s =$ # of samples needed by `L` to meet ϵ_l, δ precision
    $samples =$ sample C_s states within $merp_i$
    $examples = \{ (s, estJ(s)) : s \in samples, estJ(s) =$ `D`$(M, \hat{J}_{i-1}, merp_{i-1}, s) \}$
    $\tilde{J}_i =$ `L`(*trainset*)
    $\hat{J}_i = \{$**function** $s \rightarrow$
          $J_l, J_h =$ obtain $1 - \delta$ confidence interval on $\tilde{J}_i(s)$
          if $(J_h - J_l \leq 2\epsilon_l)$ **then** $\tilde{J}_i(s)$ **else** $\perp$ $\}$
  end for
  return $\hat{J}_i$

Figure 23: Pseudocode for the EVA algorithm. EVA generates a series of value functions, each more complete than the last, and returns a final, fully defined value function. $merp_i$ defines the expanded region for round i. `absorbp` is an indicator function which returns whether its argument is an absorbing state. The number of samples to obtain for an expansion, C_s, depends upon the approximator used (see text for details). `D` is our sampling planner. Its arguments are the MDP, a (partial) value function of known states, an admissible heuristic for unknown states, and the target state to solve. $\tilde{J}$ is the raw, approximated value function returned by the supervised learner. It is complete, but not necessarily accurate over the whole state space. $\hat{J}$ is the censored, approximated value function. It is incomplete, but accurate over the expanded region.

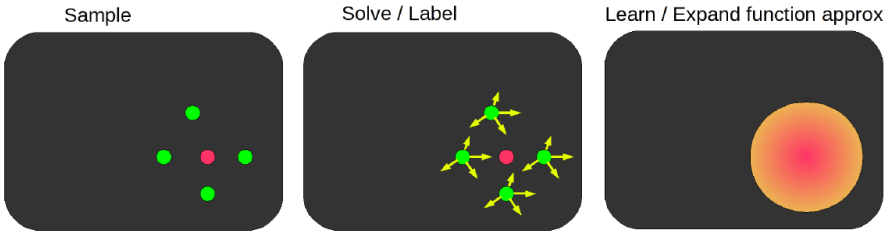

Figure 24: The basic three step expansion process. The red dot represents the solved region, pre-expansion. The larger red-orange region represents the solved region, post-expansion. Green dots represent sampled states.

function for the enlarged region. Figure 24 provides an illustration of the three basic steps. We will spend the next several subsections going over some technical details.

3.4.2.1 Component Requirements

We begin with requirements on the planning and learning components required by EVA. First, we require that the planner be able to control the accuracy of its cost-to-go estimates. Specifically, given a state s, it must be able to produce an estimate $\hat{J}(s)$ such that $|J^*(s) - \mathbb{E}[\hat{J}(s)]| \leq \epsilon_p$ for some parameter $\epsilon_p > 0$. Further, we require the planner be capable of accepting and using: (1) a partial value function of known states, (2) an admissible heuristic for unknown states, and (3) a maximum cost-to-go limit on s for early termination. This last requirement is not needed for the planner to solve a given state, but is necessary in obtaining samples within some cost-to-go range, *e.g.,* within a particular *merp*. When early termination is triggered, the planner returns $\perp$ instead of an actual estimate. Many sampling planners can meet or be made to meet these requirements. We use a custom planner called IDSS. It is based on the Sparse Sampling algorithm introduced by Kearns et al [64], and modified to meet our requirements. To meet requirement (1), IDSS treats known states as terminal states. For requirement (2), IDSS uses the heuristic value for unknown states at the leaves of the sparse lookahead tree. Meeting the last requirement requires the largest modification. To do so, we structure the planner as an iterative deepening algorithm. This enables the planner to quickly detect when the cost-to-go limit is exceeded, and terminate early. IDSS is also smart in how it builds deeper lookahead trees. It only expands those states along the greedy policy. In this respect, it can also be considered a sampled form of

real time dynamic programming [13].

The supervised learner also has requirements. We require the use of estimators which can provide confidence intervals. Further, in the limit of infinite data, the variance of the estimates must go to zero.

3.4.2.2 Determining the number of samples

Each expansion round of EVA generates a certain number of examples to train the learner. In the pseudocode for EVA, this is denoted by C_s. Let us denote the expected cost-to-go function produced by the sampling planner for some round i, as $\bar{J}_i$. Then conceptually, C_s is the number of examples the learner needs to construct an accurate (within ϵ_l with probability at least $1 - \delta$) approximation of $\bar{J}_i$ over the region of states with cost-to-go within $merp_i$. While C_s is cleanly defined, determining its value is another matter.

For some approximators, we can compute C_s directly. For example, in linear regression the $1 - \delta$ confidence interval for a given state x_0 (in feature vector form) is given by $\pm t_{\delta/2}^{n-(k+1)} s(x_0'(X'X)^{-1}x_0)^{1/2}$ where t denotes the t-distribution, n is the number of examples used in the regression, k is the number of independent variables, s is the sample standard deviation, and X is the matrix of input data without labels. By using extremum values of each feature, we can test whether tight (*i.e.,* within ϵ_l) confidence interval could be computed for all states to be approximated. C_s can then be determined by the simple online process of generating examples until the regression model can pass this test.

For others approximation methods, we rely on testing sampled states. For example, if we randomly sample 100 states and see that all yield tight confidence intervals, then we might conclude that we have obtained enough data. In practice, we use an online process of generating examples and feeding them to the approximator. When k consecutive examples can be tightly bound by the approximator, we mark the number of examples as the baseline estimate of C_s and use a multiple of that baseline as the number of examples to generate. How high a multiple to use is not known *a priori* so we must guess and check. A guess too low is equivalent to using higher ϵ_l, δ parameters than the ones given to the algorithm. This means results may exceed the theoretical error bounds. Fortunately, by doubling the

multiplier each time, we can ensure only logarithmic guesses are needed.

3.4.2.3 Sampling states within MERP

To sample states within a particular MERP, we follow the same strategy used in P-expansion: rejection sampling. In any given expansion round i, we must sample states within some cost-to-go $merp_i = i \cdot r = merp_{i-1} + r$. To determine, for a randomly sampled state s, whether it is within $merp_i$, we turn to the sampling planner. We apply the planner on s with the usual arguments: the MDP M, the value function of known (*i.e.,* previously solved) states $\hat{J}_{i-1}$, and an admissible heuristic $merp_{i-1}$. However, we also supply an early termination cost-to-go limit. If the limit is breached, s is considered outside $merp_i$ and the sample is rejected. Otherwise, a cost-to-go estimate is produced and s is accepted. We use the cost-to-go limit: $merp_i + \epsilon_p + \epsilon_{i-1}$, where ϵ_{i-1} is the precision of $\hat{J}_{i-1}$ and ϵ_p is the error introduced in the course of sample planning. In Section 3.4.3 we will derive expressions for ϵ_p and ϵ_{i-1}. For now, the important bit is that the limit reduces the search depth of the planner while ensuring that states within $merp_i$ will be accepted with high probability.

3.4.2.4 Miscellaneous Optimizations

In actual implementation we can make some practical optimizations. For example, rejection sampling requires that we use the sampling planner to compute cost-to-go estimates. If a state is accepted, we will already have its cost-to-go estimate and so it need not be computed again during labeling.

For simplicity, we have described EVA as generating a new training set each expansion round. In practice, we keep the training set from previous rounds. This provides extra examples at no additional cost.

EVA as described only terminates when $\hat{J}$ is complete. That is, when it is defined for all states. When the number of states is large, we cannot test every state to see if it is defined. Instead, we will test some number of sampled states. We also employ an additional shortcut termination condition. Note that $\hat{J}_i$ is censored, *i.e.,* made to only yield estimates for states within $merp_i$ and $\perp$ otherwise. As a result, EVA will not terminate until $merp$ has grown to cover the entire state space. In practice however, the uncensored approximated value

function, $\tilde{J}$, is often correct before *merp* matures. When this occurs, we can terminate early to save some work.

There is one other early termination case. In practice, it may be possible to specify precision parameters ϵ_l, δ which are not achievable no matter the number of training examples obtained. To prevent EVA from stalling, we will terminate early when this occurs and return the solution thus far.

Parameters ϵ_l and δ are used to compute the number of training examples to generate in an expansion round. However these parameters are not always intuitive to set. When asymptotically normal estimators are used, rather than using these two parameters, we can simply use a maximum standard deviation. The maximum standard deviation is relatively simple to understand, and equally as effective in controlling the number of training examples.

3.4.3 Theoretical Analysis

Theoretical analysis of EVA is divided into several parts. First we will establish convergence, *i.e.,* that EVA will terminate and produce a value function. Then we will prove error bounds on the value function produced. We will do so for the general case and also specifically for the case of linear regression. Our proofs assume the use of planners which only additively increase error, so we next show that our planner, IDSS, meets this additive error requirement. Finally, we will cover the planning depth required by EVA which will also help us quantify the computational complexity of EVA.

3.4.3.1 Convergence

EVA consists primarily of one for-loop. As long as each round terminates, it is easy to see that EVA must terminate. In any given round i, EVA generates a value function $\hat{J}_i$ which is defined for states within $merp_i$. Because $merp_i$ grows by r each round, in round i, $merp_i = i \cdot r$. Thus, at the end of round $i = \lceil J_{max}/r \rceil$, $\hat{J}_i$ will be defined for states within $merp_i \geq J_{max}$, which is the set of all states. Once a fully-defined value function has been constructed, the stopping condition of the for-loop is met, and EVA will end, returning $\hat{J}_i$.

The crux of convergence lies then in whether or not an expansion round always terminates. Rejection sampling, the sampling planner, and the learner are all subroutines which

are assumed to terminate. The only question is whether C_s, the number of samples needed by the learner to produce ϵ_l, δ confidence intervals for states within $merp_i$, is finite. If so, the expansion round must terminate. Recall that $\bar{J}_i$ is the expected cost-to-go function produced by the sampling planner for round i. Let us also denote an upper bound on the *limiting bias* of the learner (*i.e.,* the bias of the learner's limiting distribution) as ϵ_b. That is, in the limit of infinite data, the learner will converge to an estimate $\hat{J}$ in which $|\hat{J}(s) - \bar{J}(s)| \leq \epsilon_b$ for all states within $merp_i$. Then, as long as $\epsilon_l > \epsilon_b$, C_s will be finite. Which is to say, as long as the precision parameter is set to an achievable value, the number of samples needed to reach that precision must be finite.

In summary, EVA is guaranteed to converge so long as valid precision parameters are given.

3.4.3.2 Error bounds

The value function EVA converges to can be bounded *wrt.* the true cost-to-go function J^*. We will first analyze the general case, for any approximator capable of producing confidence intervals. Then, we will show special case analysis for linear regression.

We begin analysis of the general case by constructing an error bound on the value function for each round. Recall that round i constructs a (partial) value function $\hat{J}_i$ defined for states within $merp_i$. We will denote the upper bound on the error introduced by the planner as ϵ_p. That is, assuming no errors in its arguments, the expected value function produced by the planner $\bar{J}$, can be bounded as $|\bar{J}(s) - J^*(s)| \leq \epsilon_p$ for all states within $merp_i$. ϵ_p can be thought of as the bias of the sampling planner. Now we can introduce our error bound theorem.

Theorem 1. *In round i, for any state s within $merp_i$, $|\hat{J}_i(s) - J^*(s)| \leq i(\epsilon_p + \epsilon_l)$ with probability at least $1 - \sum_{a=0}^{i-1} C_s^a (kC_t)^{Ha}\delta$, or more loosely, $1 - C_s^i (kC_t)^{Hi}\delta$.*

$i \in \mathbb{N} \geq 0$ — *the round*

$\epsilon_p \in \mathbb{R} \geq 0$ — *error introduced by planning*

$\epsilon_l \in \mathbb{R} > \epsilon_b$ — *precision of the learner, must exceed the limiting bias of the learner*

C_s — *number of training examples used on the learner*

k — *the number of actions in the MDP*

C_t — *number of samples the sampling planner uses*

H — *planning horizon, or planning depth*

$\delta \in \mathbb{R} > 0$ — *confidence of the learner*

Proof. Proof by induction on the round i.

Base case: $i = 0$. For round 0, $i(\epsilon_p + \epsilon_l) = 0$ so $\hat{J}_0$ must be perfectly precise. Further, error probability is at most $\sum_{a=0}^{i-1} C_s^a (kC_t)^{Ha}\delta = 0$, so the precision must hold with probability 1. $\hat{J}_0$ is defined by closed form solutions and meets both these criteria.

Inductive case. We will show the bound holds for round $i+1$ assuming it held for round i. We focus on the precision bound first, and the probability of that bound holding second.

The sampling planner introduces at most ϵ_p error in constructing $\bar{J}_{i+1}$. However, the planner is called using the previous round's approximate value function, which may introduce additional error at most $i(\epsilon_p + \epsilon_l)$. Thus the error of $\bar{J}_{i+1}$ is at most their sum: $\epsilon_p + i(\epsilon_p + \epsilon_l)$.

$\hat{J}_{i+1}$ approximates $\bar{J}_{i+1}$. Given C_s examples, $\hat{J}_{i+1}$ will introduce at most ϵ_l error. Thus $\hat{J}_{i+1}$ will have error at most $\epsilon_l + \epsilon_p + i(\epsilon_p + \epsilon_l) = (i+1)(\epsilon_p + \epsilon_l)$. This meets the required precision bound of round $i+1$.

The precision bound only holds probabilistically. If the previous round's approximate value function, $\hat{J}_i$, exceeds its precision, it may jeopardizes this round's bound. This can occur with probability at most $\sum_{a=0}^{i-1} C_s^a (kC_t)^{Ha}\delta$. The planning estimate for one state relies upon, in the worst case, $(kC_t)^H$ $\hat{J}_i$ estimates — any one of which can jeopardize the state's estimate. Using union bounds, the probability of one training example exceeding its precision bound is at most $(kC_t)^H \sum_{a=0}^{i-1} C_s^a (kC_t)^{Ha}\delta$. There are C_s training examples. Using union bounds again, the probability of the training data containing one or more precision exceeding examples is at most $C_s(kC_t)^H \sum_{a=0}^{i-1} C_s^a (kC_t)^{Ha}\delta$. Finally, we must consider the learner's precision which is only probabilistically guaranteed. That precision

can be exceeded with probability at most δ.

In sum, the probability of one or more precision exceeding errors either from the training data or from the learner itself, is at most $\delta + C_s(kC_t)^H \sum_{a=0}^{i-1} C_s^a (kC_t)^{Ha}\delta = \sum_{a=0}^{i} C_s^a (kC_t)^{Ha}\delta$. Thus the probability that all precisions hold, ensuring the precision bound for round $i+1$ holding, is at least $1 - \sum_{a=0}^{i} C_s^a (kC_t)^{Ha}\delta$. This meets the required confidence for round $i+1$.

□

Given bounds on the error of each round's $\hat{J}$, we can easily derive an error bound on the output of EVA. Recall that EVA must terminate at the end of round $i = \lceil J_{max}/r \rceil$. Thus the cost-to-go function that EVA returns can be bound as follows.

Theorem 2. *EVA produces $\hat{J}$ such that for any state s, $|\hat{J}(s) - J^*(s)| \leq \lceil J_{max}/r \rceil(\epsilon_p + \epsilon_l)$ with probability at least $1 - \delta(C_s(kC_t)^H)^{\lceil J_{max}/r \rceil}$.*

Proof. Proof is evident by substituting the termination round into Theorem 1. □

The error bound we derive closely matches our intuition of expansions. First, note that the error bound increases linearly with the number of rounds. Each round has two sources of error: the planner and the learner. Because error in one expansion round may propagate to the next, each successive expansion suffers from errors it makes on top of those made in previous rounds. Fortunately, the errors are additive. Second, we note that the probability of the precision not holding, *aka.* the error probability, increases multiplicatively. This is because the number of accurate cost-to-go estimates needed to compute the value for a given state increases exponentially with the cost-to-go of that state. A state with high cost-to-go must occur in a later round. That state's estimate may only be accurate if the values it depends upon are accurate — values computed in the previous round. Those values in turn are only accurate if values computed in the twice previous round are accurate, and so on. The key is that each state's estimation may rely upon many values from the previous round. As a result, one gets an exponential explosion in the number of previously estimated values needed to compute an accurate estimate. Although the error probability increases exponentially with the number of rounds, we note that for asymptotically normal estimators,

we can reduce δ exponentially fast by increasing C_s the number of training examples given to the learner. Specifically, δ drops according to $O(\text{erfc}(\sqrt{C_s}))$ which can be bounded by $O(e^{-C_s})$. Thus, for this class of estimators, the exponential rise in error probabilities can be easily combated. MDPs which require more rounds will require linearly more examples per expansion.

Using our error bound, we can prove one additional guarantee.

Theorem 3. *Given an approximator that can represent J^*, EVA will converge to J^*.*

Proof. By Theorem 2 EVA produces a value function, $\hat{J}$, in which for any state s, $|\hat{J}(s) - J^*(s)| \leq \lceil J_{max}/r \rceil(\epsilon_p + \epsilon_l)$ with probability at least $1 - \delta(C_s(kC_t)^H)^{\lceil J_{max}/r \rceil}$.

Given that the approximator can represent J^*, by increasing C_s, the number of examples given to the learner, we can achieve values of ϵ_l and δ arbitrarily close to zero. Further, we can also set ϵ_p, the precision of the planner, arbitrarily close zero by increasing the sampling fidelity of the planner. As ϵ_p, ϵ_l, and δ approach zero, the error incurred by $\hat{J}$ for any given state will approach zero with probability approaching 1.

Thus as the amount of sampling increases, EVA will converge to J^*. □

Now we will consider a special case bound for a particular learning algorithm: linear regression. First note that linear regression is an asymptotically normal estimator and so our general case results apply. However, by taking advantage of the special characteristics of using a linear model, we can obtain a better error bound.

Theorem 4. *In round i, for any state s within $merp_i$, $|\hat{J}_i(s) - J^*(s)| \leq i(\epsilon_p + \epsilon_l)$ with probability at least $1 - i\delta$.*

Proof. The proof for the linear case is the same as that of the general case excepting the growth of the error probability. Recall that we use an inductive proof over the round i. In the inductive step, we aim to show the bound holds for round $i + 1$ assuming it holds for round i. In the general case proof we relied upon the union bound to obtain the probability of having one or more precision exceeding errors in the training data. This resulted in a bound, $C_s(kC_t)^H\delta_i$, where δ_i denotes the error probability of the previous

round. In linear regression, the union bound is not necessary. The probability that all $C_s(kC_t)^H$ state estimates fall within the requisite precision is equivalent to the probability that the coefficients of the linear model learned in round i fall within some precision. This is because all the state estimates come from *the same* linear model — the coefficients are only computed once and used to determine every state estimate. The only union bound needed is the one to combine the probability of error from previous rounds with the probability of error from the learner of the current round. As a result, the error probability grows additively: $\delta_{i+1} = \delta_i + \delta$. Since $\delta_0 = 0$, for any round i, the error probability is at most $i\delta$.

□

3.4.3.3 IDSS Precision

So far we have assumed that IDSS can compute cost-to-go estimates with additive error. In this section, we will prove that it does. Since IDSS is based on Sparse Sampling, our proof will follow suite. First, we will derive error bounds when a (partial) value function of known estimates is used with Sparse Sampling. We call this algorithm SS+. It demonstrates the additive nature of the errors. Then we will complete the transformation to IDSS, adding in the maximum cost-to-go limit and admissible heuristic, and show that it maintains the error bounds.

We begin with a statement of the main result of SS+, followed by requisite lemmas.

Theorem 5. *SS+ has the following behavior:*

Given: (1) an MDP with k actions, (2) (partial) cost-to-go function Ω *accurate within* ϵ*, (3) input state* $s \in S$*, and (4) parameters* $\lambda \in (0, G_{max}/\gamma)$ *denoting the sampling precision,* $C_t \in \mathbb{Z}$ *the number of times to sample a state-action transition, and* $H \in \mathbb{Z}$ *the planning depth.*

Output: an estimate of the cost-to-go of state s: $J^H(s)$

Satisfying:

[Concentration Bound]: $|J^*(s) - J^H(s)| \leq \alpha_H$ *with probability* $\geq 1 - \beta_H$*, where:*

$$\alpha_H = \sum_{i=1}^{H} \gamma^i \lambda + \gamma^H J_{max} + \epsilon$$

$$\beta_H = (kC_t)^H (ke^{-\lambda^2 C_t / J_{max}^2})$$

[Precision Bound]: $|J^*(s) - \mathbb{E}[J^H(s)]| \leq (1-\beta_H)\alpha_H + \beta_H J_{max}$

[Variance Bound]: $Var[J^H(s)] \leq \frac{1}{4}J_{max}^2$

Proof. We will prove each part separately. Our proof will rely heavily on Lemma 2 which bounds $|Q^*(s,a) - Q^H(s,a)|$. We will cover Lemma 2 after this theorem. For now, we ask the reader to take the result on faith.

Concentration Bound: from Lemma 2 it follows that $|Q^*(s,a) - Q^H(s,a)| \leq \alpha_H$ with probability at least $1 - (kC_t)^H e^{-\lambda^2 C_t / J_{max}^2}$.

$$\begin{aligned}
|J^*(s) - J^H(s)| &= |min_a Q^*(s,a) - min_a Q^H(s,a)| \\
&\leq max_a |Q^*(s,a) - Q^H(s,a)| \\
&\leq max_a(\alpha_H) \\
&\leq \alpha_H
\end{aligned}$$

Since the precision bound for J^H requires Q^H to fall within its bound for all k actions, it holds with probability at least $1 - (kC_t)^H k e^{-\lambda^2 C_t / J_{max}^2} = 1 - \beta_H$.

Precision Bound: Since $|J^*(s) - J^H(s)| \leq \alpha_H$ with probability at least $1 - \beta_H$, we can easily bound $\mathbb{E}[J^H(s)]$. With probability $1 - \beta_H$, the absolute error is within α_H. With probability β_H the absolute error is unknown but certainly within J_{max}. Thus:

$$|J^*(s) - \mathbb{E}[J^H(s)]| \leq (1-\beta_H)\alpha_H + \beta_H J_{max}$$

Variance Bound: A bound on the variance is derived by taking advantage of the fact that cost-to-go estimates are bounded between 0 and J_{max}.

$$\begin{aligned}
Var[J^H(s)] &= E[J^H(s)^2] - E[J^H(s)]^2 \\
&\leq E[J_{max} J^H(s)] - E[J^H(s)]^2 \\
&\leq (J_{max} - E[J^H(s)])E[J^H(s)] \\
&\leq (\frac{1}{2}J_{max})(\frac{1}{2}J_{max}) \\
&\leq \frac{1}{4}J_{max}^2
\end{aligned}$$

□

Now we will move to prove Lemma 2 which bounds $|Q^*(s,a) - Q^H(s,a)|$. First however, we will need to bound the error from using a sample mean instead of the expectation.

Lemma 1. *Let* $U^*(s,a) = G(s,a) + \gamma \frac{1}{C_t} \sum_{i=1}^{C_t} J^*(s_i)$.

Then for all state-action pairs (s,a)*, with probability at least* $1 - e^{-\lambda^2 C_t / J_{max}^2}$*, we have*

$$|Q^*(s,a) - U^*(s,a)| = \gamma |\mathbb{E}[J^*(T(s,a))] - \frac{1}{C_t} \sum_{i}^{C_t} J^*(s_i)| \leq \lambda$$

where s_i *is drawn from* $T(s,a)$*, and* $\lambda > 0$ *is a parameter controlling the desired sampling precision.*

Proof. This lemma directly corresponds to Lemma 3 in [64]. Its proof is immediate from the Chernoff bound.

□

To bound $|Q^*(s,a) - Q^H(s,a)|$, we need to consider how errors from using a sample mean propagate over multiple horizons. Recall that Ω is the (partial) value function of known values given to SS+.

Lemma 2. *Let us define* $Q^n(s,a)$ *as the horizon-n Q function using sample means. Similarly, we define* $J^n(s)$ *as the horizon-n cost-to-go function using sample means. Formally:*

$$Q^n(s,a) = G(s,a) + \gamma \frac{1}{C_t} \sum_{i=1}^{C_t} J^{n-1}(s_i)$$

$$Q^0(s,a) = 0$$

$$J^n(s) = \begin{cases} min_a Q^n(s,a) & : \Omega(s) = \perp \\ \Omega(s) & : else \end{cases}$$

where $n \in \mathbb{N}$ *and* s_i *are samples from* $T(s,a)$.

Then for all $n \geq 0$*, with probability at least* $1 - (kC_t)^n e^{-\lambda^2 C_t / J_{max}^2}$*, we have*

$$|Q^*(s,a) - Q^n(s,a)| \leq \alpha_n$$

$$\alpha_0 = J_{max}$$

$$\alpha_n = max(\sum_{i=1}^{n} \gamma^i \lambda + \gamma^n J_{max}, \epsilon)$$

where $0 \leq \lambda \leq G_{max}/\gamma$ *is the sampling precision parameter and* $C_t > 0$ *is a parameter dictating the number of times to sample a state-action transition.*

Requiring $\lambda \leq G_{max}/\gamma$ ensures a minimum level of precision. It also ensures that $\sum_{i=1}^{n} \gamma^i \lambda + \gamma^n J_{max}$ is monotonically non-increasing.

Proof. We will prove by induction on the horizon length n. We will focus on precision bound first, and probability of that precision bound holding second.

Precision, base case: $n = 0$

$$\begin{aligned} |Q^*(s,a) - Q^0(s,a)| &= |Q^*(s,a) - 0| \\ &\leq |J_{max} - 0| \\ &\leq J_{max} = \alpha_0 \end{aligned}$$

Precision, inductive case: we bound the total error by splitting it into error from sampling and error from using a finite horizon. The former is bounded by Lemma 1. The latter

requires that we track how error accumulates over multiple horizons.

$$
\begin{aligned}
|Q^*(s,a) - Q^{n+1}(s,a)| &= \gamma|\mathbb{E}_{s'}[J^*(s')] - \frac{1}{C_t}\sum_i J^n(s_i)| \\
&\leq \gamma(|\mathbb{E}_{s'}[J^*(s')] - \frac{1}{C_t}\sum_i J^*(s_i)| + |\frac{1}{C_t}\sum_i J^*(s_i) - \frac{1}{C_t}\sum_i J^n(s_i)|) \\
&\leq \gamma(\lambda + |\frac{1}{C_t}\sum_i J^*(s_i) - \frac{1}{C_t}\sum_i J^n(s_i)|) \\
&\leq \gamma(\lambda + \frac{1}{C_t}\sum_i |J^*(s_i) - J^n(s_i)|) \\
&\leq \gamma(\lambda + \frac{1}{C_t}\sum_i max(|min_a Q^*(s_i,a) - min_a Q^n(s_i,a)|, \epsilon) \\
&\leq \gamma(\lambda + \frac{1}{C_t}\sum_i max(max_a(|Q^*(s_i,a) - Q^n(s_i,a)|), \epsilon) \\
&\leq \gamma(\lambda + \frac{1}{C_t}\sum_i max(\sum_{i=1}^{n}\gamma^i\lambda + \gamma^n J_{max}, \epsilon) \\
&\leq \gamma(\lambda + max(\sum_{i=1}^{n}\gamma^i\lambda + \gamma^n J_{max}, \epsilon)) \\
&\leq \gamma(\lambda + \alpha_n)
\end{aligned}
$$

We now continue analysis depending on ϵ. If $\epsilon < \sum_{i=1}^{n}\gamma^i\lambda + \gamma^n J_{max}$, then

$$
\begin{aligned}
&\leq \gamma(\lambda + \sum_{i=1}^{n}\gamma^i\lambda + \gamma^n J_{max}) \\
&\leq \sum_{i=1}^{n+1}\gamma^i\lambda + \gamma^{n+1}J_{max} \\
&\leq max(\sum_{i=1}^{n+1}\gamma^i\lambda + \gamma^{n+1}J_{max}, \epsilon) = \alpha_{n+1}
\end{aligned}
$$

Otherwise, $\epsilon \geq \sum_{i=1}^{n} \gamma^i \lambda + \gamma^n J_{max}$:

$$
\begin{aligned}
&\leq \gamma(\lambda + \epsilon) \\
&\leq \gamma\lambda + \epsilon - (1-\gamma)\epsilon \\
&\leq \gamma\lambda + \epsilon - (1-\gamma)(\sum_{i=1}^{n} \gamma^i \lambda + \gamma^n J_{max}) \\
&\leq \gamma\lambda + \epsilon - (1-\gamma)\frac{\lambda}{1-\gamma} \\
&\leq \gamma\lambda + \epsilon - \gamma\lambda \\
&\leq \epsilon \\
&\leq max(\sum_{i=1}^{n+1} \gamma^i \lambda + \gamma^{n+1} J_{max}, \epsilon) = \alpha_{n+1}
\end{aligned}
$$

This concludes the precision half of our proof. We now tackle the probability of the precision bound holding. We will prove a tighter bound, showing that the probability of error greater than α_n, is bounded by

$$B^n \leq \sum_{i=0}^{n-1} (kC_t)^i e^{-\lambda^2 C_t / J_{max}^2}$$

For the base case, $n = 0$, the precision bound stems simply from the definition of Q^0 without use of any probabilistic simplifications so the probability of error $B^0 = 0$. Clearly, this satisfies our requisite upper bound.

We now turn our attention to the inductive case, on B^{n+1}, the probability that the precision bound for Q^{n+1} will hold. The simplification from $|\mathbb{E}_{s'}[J^*(s')] - \frac{1}{C_t}\sum_i J^*(s_i)|$ to λ has error probability bounded by $e^{-\lambda^2 C_t / J_{max}^2}$. The simplification from $|J^*(s_i) - J^n(s_i)|$ to α_n for up to C_t sampled states, has error probability bounded by $kC_t B^n$. The error bound

for B^{n+1} can thus be bounded by a sum of these two error sources.

$$\begin{aligned} B^{n+1} &\leq e^{-\lambda^2 C_t/J_{max}^2} + kC_t B^n) \\ &\leq e^{-\lambda^2 C_t/J_{max}^2} + kC_t B^n \\ &\leq e^{-\lambda^2 C_t/J_{max}^2} + kC_t \sum_{i=0}^{n-1} (kC_t)^i e^{-\lambda^2 C_t/J_{max}^2} \\ &\leq \sum_{i=0}^{n} (kC_t)^i e^{-\lambda^2 C_t/J_{max}^2} \end{aligned}$$

□

Now that we have proved an error bound for SS+, we must show that IDSS which adds a maximum cost-to-go limit and admissible heuristic will maintain the error bound. Specifically in the context of EVA:

1. We assume an admissible heuristic, *merp*. Further, we require that the cost-to-go function of known values is defined for all states within *merp*.

2. We take as an additional argument, a maximum cost-to-go limit $maxJ = merp + r + \alpha_H$.

Recall that in order to terminate when the maximum cost-to-go limit is exceeded, IDSS has been modified to be an iterative deepening algorithm. If $J^H(s)$ for some horizon H exceeds $maxJ$, IDSS will return $\perp$. Otherwise, the algorithm will return $J^H(s)$ as SS+ normally would.

Theorem 6. *If $J^*(s) \leq merp + r$ then IDSS will return $J^H(s)$ as SS+ normally would with probability at least $1 - \beta_H$.*

Proof. We derive upper bound for error probability.

Let A be the logical expression that $J^*(s) \leq merp + r$.

Let B be the logical expression that IDSS returns $\perp$.

$$
\begin{aligned}
P(A \rightarrow \neg B) &= 1 - P(A \wedge B) \\
&= 1 - P(B|A)P(A) \\
&\geq 1 - P(B|A) \\
&\geq P(\neg B|A) \\
&\geq P(J^H(s) \leq merp + r + \alpha_H \mid A) \\
&\geq P(J^H(s) - J^*(s) \leq merp + r + \alpha_H - J^*(s) \mid A) \\
&\geq P(J^H(s) - J^*(s) \leq \alpha_H \mid A) \\
&\geq P(|J^H(s) - J^*(s)| \leq \alpha_H \mid A) \\
&\geq (1 - \beta_H)
\end{aligned}
$$

Note that we can convert $\neg B$ to $(J^H(s) \leq merp+r+\alpha_H)$ because $J^n(s)$ is monotonically increasing and α_n is monotonically non-increasing *wrt.* n. So if it's true for H it must be true for all $0 \leq n \leq H$. □

3.4.3.4 Planning Depth

In the previous section, we saw in analysis of SS+ that the planning depth is a parameter which, along with sampling parameters, control the error incurred by the planner. We also saw that IDSS is guaranteed to produce the same behavior with SS+ with high probability. What we have not yet covered is that IDSS holds an important advantage, it has a smaller effective horizon. The addition of an admissible heuristic combined with a cost-to-go limit enables the planner to search less deeply while maintaining the same error bounds.

First we need the concept of stochastic locality. This is the idea that for a given state action pair, the next states are close together in terms of their cost-to-gos. In other words, a step cannot transport you very far away from where you expect to be. Stochasticity in physical systems, for example, follows this behavior. Consider a simple, wheeled robot. While there may be some uncertainty in position after sending a command to move forward 1 meter, that uncertainty is relatively concentrated around the 1 meter ahead location. One

would not expect the robot to be 10 meters behind where it started, or to suddenly be 10 meters ahead. Formally we define stochastic locality by $\mathbb{E}[J^*(T(s,a))] - \min_{s'} J^*(s')$ where $s' \sim T(s,a)$.

Now we can introduce a bound on the effective horizon of IDSS.

Theorem 7. *If stochastic locality is bounded within C_l, and the horizon parameter is set to H, then the expected effective horizon is bounded by $\frac{H(1-\gamma)}{\gamma(1-\gamma^H)}[(r + C_l)/c_{min} - 1]$.*

Proof. Let us suppose we must estimate the cost-to-go of a state s_i with partial value function Ω defined for states within *merp*, admissible heuristic *merp*, and cost-to-go limit $merp + r$. r represents the radius of IDSS. If state s_i is within *merp* IDSS will solve it using Ω directly and the effective horizon will be zero. Thus we will focus on the case when $J^*(s) > merp$.

We begin by noting that IDSS stops if $\hat{J}(s_i)$ ever exceeds $merp + r$. As a result, the effective horizon of IDSS for a state with cost-to-go $merp + r$ is lower than the effective horizon for that state without IDSS' early termination condition. Let us call this variation, IDSS$^-$. States with higher cost-to-gos are similarly limited. Thus the effective horizon of IDSS is limited by the effective horizon of IDSS$^-$. The rest of this proof will focus on obtaining an expression for this upper bound.

We first introduce a random variable Z, denoting the number of steps taken from s_i, such that the state we end up in after taking these steps, s_f, is the first state whose value, $J^*(s_f) \leq merp$. Z is the effective horizon for s_i under IDSS$^-$.

$$\begin{aligned}
J^*(s_i) &= \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i G_i] \\
&= \mathbb{E}[\sum_{i=0}^{Z} \gamma^i G_i + \sum_{i=Z+1}^{\infty} \gamma^i G_i] \\
&= \mathbb{E}[\sum_{i=0}^{Z} \gamma^i G_i] + \mathbb{E}[\sum_{i=Z+1}^{\infty} \gamma^i G_i] \\
&\geq \mathbb{E}[\sum_{i=0}^{Z} \gamma^i G_i] + (merp - C_l)
\end{aligned}$$

However $J^*(s_i)$ must be within $merp + r$. Thus:

$$\mathbb{E}[\sum_{i=0}^{Z} \gamma^i G_i] + (merp - C_l) \leq merp + r$$
$$\mathbb{E}[\sum_{i=0}^{Z} \gamma^i G_i] \leq r + C_l$$
$$\mathbb{E}[\sum_{i=0}^{Z} \gamma^i G_{min}] \leq r + C_l$$
$$\mathbb{E}[\sum_{i=0}^{Z} \gamma^i] \leq (r + C_l)/G_{min}$$
$$\mathbb{E}[\frac{1 - \gamma^{Z+1}}{1 - \gamma}] \leq (r + C_l)/G_{min}$$

We define random variable $X = \frac{1-\gamma^{Z+1}}{1-\gamma}$, so we can rewrite that $\mathbb{E}[X] \leq (r + C_l)/c_{min}$. Note that Z has range $[0, H]$, and that X has range $[1, \frac{1-\gamma^{H+1}}{1-\gamma}]$. Further note that $Z = ln(1 - X(1-\gamma))/ln(\gamma) - 1$. This means that Z is monotonically increasing, and that Z is convex since $1 - X(1-\gamma)$ is monotonically decreasing to zero (within the domain of X). Because of these properties, we can create a linear upper bound for Z which we will call Z':

$$Z \leq Z' = \frac{ln(\gamma^{H+1})/ln(\gamma) - 1}{(1 - \gamma^{H+1})/(1-\gamma) - 1}(X - 1)$$
$$= \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}(X - 1)$$

We can then bound the expectation of Z as:

$$Z \leq \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}(X - 1)$$
$$Z \leq \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}X - \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}$$
$$\mathbb{E}[Z] \leq \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}\mathbb{E}[X] - \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}$$
$$\leq \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}\frac{r + C_l}{c_{min}} - \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}$$
$$\leq \frac{H(1-\gamma)}{\gamma(1 - \gamma^H)}[(r + C_l)/c_{min} - 1]$$

□

3.4.3.5 EVA Complexity

EVA, over the course of i rounds, will sample iC_s states and make as many calls to the sampling planner. It will also make i calls to the supervised learner. The cost of sampling states is linear in the number of states sampled. If we also treat the calls to the planner and learner as constant operations then the overall complexity of EVA is $O(iC_s + i) = O(iC_s)$. This is linear in the number of rounds and in the number of examples required for training.

However, calls to the planner and learner are generally not constant time operations. The learner is usually polynomial in the number of training examples. The planner is generally exponential in the planning depth. Since there are fewer calls made to the learner, and its complexity is lower than that of the planner, it is not hard to see that the complexity of EVA is dominated by the cost of iC_s calls to the planner.

In the worst case, the depth of the planner can be the horizon of the MDP: $H = log_\gamma(\epsilon_p(1-\gamma))$. In practice however, most domains exhibit various degrees of stochastic locality which lowers the effective planning depth to $\frac{H(1-\gamma)}{\gamma(1-\gamma^H)}[(r + C_l)/c_{min} - 1]$. Assuming the planner still dominates the complexity, this translates to total complexity:

$$iC_s b^{\frac{H(1-\gamma)}{\gamma(1-\gamma^H)}[(r+C_l)/c_{min}-1]}$$

where b is the branching factor of the planner. In the worst case, b is bounded by kC_t where k is the number of actions in the MDP and C_t is the number of times a state-action transition is sampled.

3.4.4 Empirical Studies

Our experiments are broken up into two parts. In the first, we will focus on gaining a better understanding of EVA and its benefits *wrt.* other approximation methods. These experiments will be on simple domains where we can easily visualize the results. We compare against LSPI, a well known and state of the art function approximation algorithm which uses a linear model. In the second part, we will explore how EVA scales as the size of the state space increases. There, we will compare against standard tabular methods.

3	4	4	4	5	5	5	6	6	6
3	3	4	4	4	5	5	5	6	6
3	3	3	4	4	4	5	5	5	6
2	3	3	3	4	4	4	5	5	5
2	2	3	3	3	4	4	4	5	5
2	2	2	3	3	3	4	4	4	5
1	2	2	2	3	3	3	4	4	4
1	1	2	2	2	3	3	3	4	4
1	1	1	2	2	2	3	3	3	4
	1	1	1	2	2	2	3	3	3

Figure 25: Progression of EVA in a simple 10x10 grid-world. Each cell is labelled with the expansion round in which its cost-to-go is computed. The cell at the bottom left hand corner is the terminal state and initially known.

3.4.4.1 Understanding EVA

To gain a better understanding of how EVA works, we study it in three simple domains. The first is a simple 10x10 grid-world with a single terminal state in the bottom left at (0,0). Rewards are uniformly -1. The second is a similar grid-world but divided by obstacles into two rooms. The third is also a two room domain but running into a wall results in having to restart from the bottom right cell. In these examples, EVA uses radius $r = 3$, planner precision $\epsilon_p = 0.001$, and a linear regression as the learner. This lets us perform an appropriate comparison with LSPI which is limited to linear models. Note that rather than using the standard ϵ_l, δ parameters to specify the number of examples to generate per expansion round, we use the alternative maximum standard deviation parameter: $\sigma_{max} = 0.001$. Precision for LSPI is set to 0.001.

Figure 25 shows how EVA progress on the simple 10x10 grid-world. The state variables used are the `X` and `Y` coordinates of the agent. From the first expansion round, EVA produces an accurate value function. Later rounds yield no changes to the linear model. In practice, EVA would terminate after the first round. However, for illustrative purposes we run all six rounds. To compare against LSPI, we use the following state-action variables: post-action `X`, post-action `Y`, and a dummy variable indicating whether the current state is a terminal state. This set of variables enables the linear model to capture the true value function. In

4	4	4	5	5	5	6	6	6	7		7	8	8	8	9	9	9	10	10	10
3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8	9	9	9	10	10
3	3	4	4	4	5	5	5	6	6		7	8	8	8	9	9	9	10	10	10
3	3	3	4	4	4	5	5	5	6		8	8	8	9	9	9	10	10	10	11
2	3	3	3	4	4	4	5	5	5		8	8	9	9	9	10	10	10	11	11
2	2	3	3	3	4	4	4	5	5		8	9	9	9	10	10	10	11	11	11
2	2	2	3	3	3	4	4	4	5		9	9	9	10	10	10	11	11	11	12
1	2	2	2	3	3	3	4	4	4		9	9	10	10	10	11	11	11	12	12
1	1	2	2	2	3	3	3	4	4		9	10	10	10	11	11	11	12	12	12
1	1	1	2	2	2	3	3	3	4		10	10	10	11	11	11	12	12	12	13
	1	1	1	2	2	2	3	3	3		10	10	11	11	11	12	12	12	13	13

Figure 26: Progression of EVA in a two-room grid-world. Each cell is labelled with the expansion round in which its cost-to-go is computed. The cell at the bottom left hand corner is the terminal state and initially known.

our experiments, LSPI also performs well, consistently producing an optimal policy.

Figure 26 shows how EVA progress on the two-room domain. The state variables used are the `X` and `Y` coordinates of the agent, a dummy variable of whether the agent is in the right room and below the doorway, and the `Y` coordinate of the agent interacted with the dummy variable. In this domain, EVA's first expansion round is not accurate across the whole domain but only within the first (left) room. Thus, EVA methodically expands outward. Each round yields no changes to the linear model until expansion enters the second room. At that point the model is updated to accommodate the second room, and the learned linear model becomes accurate over the whole state space. EVA would normally terminate after the first expansion in the second room, however, for illustrative purposes we show EVA running all 13 rounds.

To compare LSPI in this domain we use the state-action variable set (X',Y',R,RY,1) where X' and Y' are the post-action coordinates, R is a dummy variable indicating whether the agent is in the right room below the door, and Y is the pre-action `Y` coordinate. This set of variables enables the linear model to capture the true value function. In this domain, LSPI fares well again although there are some variations in the results. While most runs yield a high performing policy, a few would yield policies in which one could not escape the

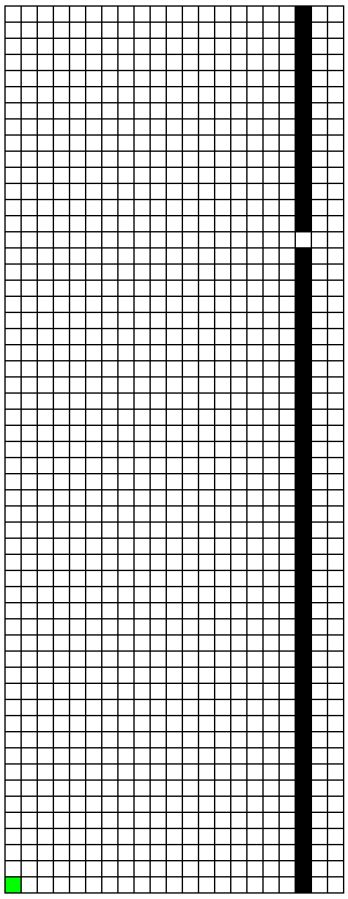

Figure 27: A large two-room grid-world in which the doorway between the rooms is not centered. This domain illustrates how LSPI can have difficulty producing good solutions even when the optimal Q function can be perfectly represented.

right room. These policies appeared more often when initial weights were set randomly than when they were set to zero. As is the case with all bootstrapping methods, an unlucky choice of initial weights could lead to a sequence of policies whose value functions are difficult to approximate, resulting in suboptimal solutions. The zero initial weight in this case, appears to be a better starting point than a randomly chosen weight. In practice, by running LSPI a few (*e.g.,* 10) times and taking the best policy produced, we were able to consistently obtain good results.

Figure 27 shows our third domain. It is a larger two-room grid-world, but the doorway is not centered and running into a wall results in having to restart from the bottom right cell. EVA performs identically in this domain as in the first two-room domain. LSPI, however performs poorly. While the policy in the first (left) room is consistently accurate, the

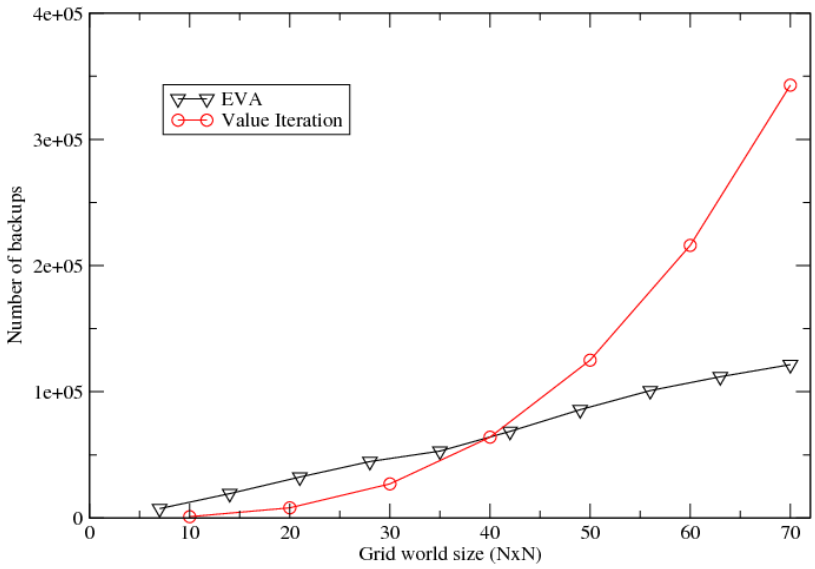

Figure 28: Scaling comparison of EVA and VI.

policy in the right room is not. In 100 runs of LSPI, none produced an optimal policy. This behavior did not change when we increased the number of samples from 1,000 to 50,000. Similarly, changing how we obtained the data had no impact. We tested both uniform random sampling as well as sampling random trajectories of varying lengths. Changing the initialization of the weight vector from zero weights to random weights also had no impact. In sum, unless we started with the correct weights, LSPI would produce suboptimal results. This example highlights the benefit of EVA's expansion approach. Despite a model that can exactly represent the true value function, iterative methods such as LSPI cannot converge to it.

3.4.4.2 Scaling

From the complexity analysis of EVA, we know it scales linearly with the number of rounds. Since each round increases the coverage of the approximation by radius r, the number of rounds is bound by $\lceil J_{max}/r \rceil$. Thus, all else being equal, EVA will scale linearly with J_{max} — with the horizon of the problem. Importantly, this is non-dependent of the size of the state space. That is, the complexity depends upon characteristics that may be influenced by state space size, but not on the state space size.

Figure 28 confirms our analysis. As we scale up the size of the grid-world test domain,

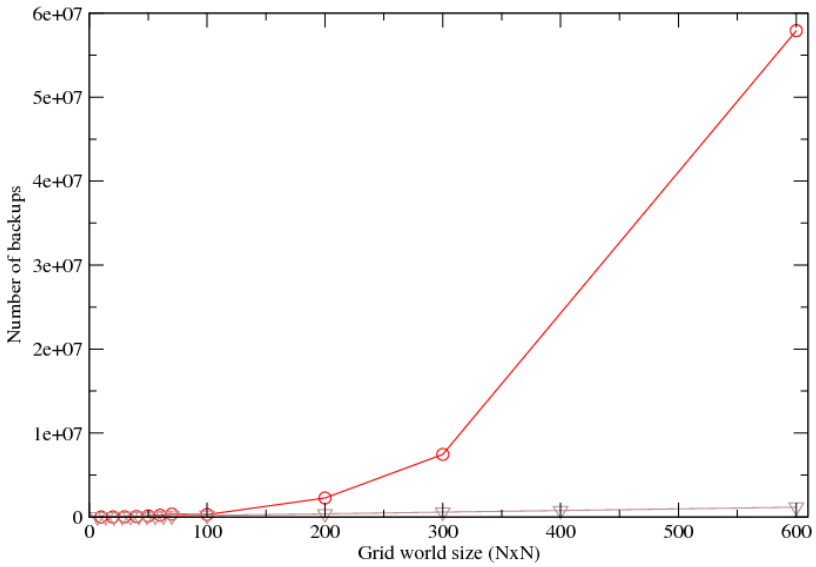

Figure 29: Extended scaling comparison of EVA and VI.

the number of states increases quadratically while the depth of the problem only increases linearly. As we see, EVA exactly follows the linear scaling — it is only affected by problem depth. By contrast, VI scales polynomially because it is tied to the size of the state space.

As the size of the grid-world continues to grow, we see in Figure 29 that the computational cost of our tabular method quickly outstrips that of EVA. By the time the size of the state space reaches 100k, the computational cost of EVA is over an order of magnitude lower than that of VI.

3.5 Summary

In this chapter, we tackled the problem of building an effective approximation algorithm with good convergence properties. It showed that the key difficulty that arises in approximation is the use of bootstrapping. This observation led to an alternative, expansion-based approach, which constructs a series of value functions that cover ever larger portions of the state space. A key property of the expansion approach is that it avoids bootstrapping. Instantiating the expansion approach comes with its own set of difficulties and we presented two different strategies for addressing these difficulties.

The first expansion strategy led us to develop (A)RVI, which makes a compromise between expansion and bootstrapping. It is fast, easy to understand, and very effective

under tabular representations. Under approximate representations, it turns out to be a reinvention of the idea behind two algorithms — Grow-Support and ROUT [18] which have convincing empirical results. Unfortunately, the strategy can only be used in acyclic domains. Further, it is not guaranteed to converge to an optimal or near-optimal policy.

Our second strategy turns to the use of sampling planners and approximators that can provide confidence intervals. This strategy lets us avoid bootstrapping completely. Importantly, it is amenable to theoretical analysis. I first showed that the corresponding algorithm, EVA, converges. I then showed that the solution it produces can be probabilistically bounded *wrt.* the optimum. The bound is well behaved: its precision depends upon the precision of the planner and learner used, and grows linearly with the number of expansion rounds. The probability of exceeding the precision depends upon the number examples used to train the learner and, in the worst case, grows exponentially in the number of expansion rounds. Fortunately, this worst case translates to only a linear increase in the number of examples needed to train the learner in each expansion round, assuming the use of an asymptotically normal estimator. Further, for the special case of linear regression, the probability of exceeding the precision only grows linearly in the number of expansion rounds. This linear growth translates to just a logarithmic increase in the number of examples needed to train the learner in each expansion round.

The price we pay for these results is an increase in computational complexity compared to approximate solutions such as LSPI. In the worst case, for fully connected MDPs in which every state-action pair can transition to all states, EVA's cost is exponential in the horizon of the MDP. Fortunately, in many real-world problems, stochasticity tends to be more localized. We introduced a metric for measuring this "stochastic locality", and showed that locality significantly limits the effective horizon required. In empirical studies, we see that EVA performs well on a variety of domains.

Our two expansion strategies provide different strengths. EVA has good theoretical guarantees but is potentially exponential in the horizon of the MDP. It works best on domains with short horizons or those with localized stochasticity. ARVI has no such exponential ties, but its theoretical guarantees are worse and its applicability is limited to

acyclic domains.

CHAPTER IV

LEVERAGING HUMAN INPUT

At the end of the day, the pure MDP formulation is a difficult one. Real world problems can be so complex that they cannot be practically solved even with convergent approximation methods due to the number of samples needed. A pure MDP formulation also attempts to solve the problem from scratch — an approach that is rarely taken in human and animal problem solving. It behooves us to develop ways to leverage additional inputs and augment the MDP formalism.

This chapter focuses on leveraging a very promising source of additional input, those from humans. Human help is rich. People are full of common-sense knowledge. They typically also have a wealth of contextual knowledge about the domain and can often provide deep insights into the problem. Further, human help is often nearby and readily available. Consider common applications such as robotic assistants in homes or administrative assistants on computers. In these and other applications, humans are the end-users, making them not only a readily available source of information but one that is also well motivated.

Research in leveraging human input to help solve decision problems dates back several decades. Work from as early as 1992 built automatic controllers for flying planes by observing and imitating human pilots [108]. Since then, many different approaches for leveraging human input has been explored. The automatic pilot example falls under the approach of learning from demonstration [6] or imitation learning [104], both of which rely upon the availability of examples of appropriate behavior. Alternative methods such as learning from advice [5] rely upon a source of critiques and suggestions. Still others, such as reward shaping [36, 106] methods, focus on obtaining performance improvements by augmenting the reward function to reduce sparsity and guide exploration. Over the years, research in leveraging human input has achieved some impressive results such as helicopter flying [93], and teaching a AIBO robot basic soccer skills [51].

While much progress has been made, and many methods for leveraging human input have been explored, relatively little research has been focused on addressing the unique characteristics of the types of human input we are most likely to encounter — that of end-users.

Leveraging help from end-users is nontrivial. End-users may know little of artificial intelligence, typically have never taught a machine, can be expected to have only limited time available to help, often give noisy or ambiguous inputs, and often focus on suboptimal, satisficing solutions rather than optimal ones. It would be inaccurate to treat end-users as a kind of optimal or near-optimal oracle. The solicitation and use of input from end-users must also be considered. Care must be taken to ensure engagement of the end-user, lest they lose interest and walk away, to maximize the use of end-users' limited time, and to limit the impact of potential errors and suboptimalities.

Recent works have begun to address these issues. These works explore different schemes for leveraging human input and validate directly through user studies, allowing us to ground a discussion of effectiveness and applicability. For example, Isbell et al. [61] studied a reinforcement learning agent in a long-term social environment and showed how user feedback could be inconsistent and tended to drift over time. Thomaz et al. [131] studied reward shaping and showed that people would use the reward channel not only for feedback on actions performed, but also for future-directed guidance and to provide motivation. Still others such as Knox et al. [67] have studied how to combine user feedback with well known reinforcement learning methods such as Q-Learning [140] to improve learning speed without sacrificing solution quality.

In this chapter, I continue this line of inquiry. Specifically, I focus on two issues: (1) how to leverage input so as to insulate against errors, and (2) how to solicit input from end-users. In line with my intended audience of end-users, my work focuses on the learning from demonstration (LfD) [6] context. LfD typically does not require explicit knowledge, instead it focuses on task performance thus allowing end-users to make use of their tacit knowledge [103], *i.e.,* intuitive knowledge difficult to verbalize or formalize. LfD also does not require familiarity of machine learning on the part of the teacher. Finally, LfD appears

effective, leading to a wide range of successes [93, 4].

In the following sections, I will cover issue (1) first. My guiding insight and general approach for leveraging human input is to treat it as a source of scaffolding rather than as a source of direct answers. Scaffolding is a concept from educational psychology in which the teacher plays the supportive role of enabling the learner to achieve something they would not be able to accomplish otherwise. In this context, using human input for scaffolding translates to extracting, from human input, hints, tips, and other knowledge on *how* to solve the decision problem so that the machine can efficiently compute the solution. Because scaffolding consists of supportive knowledge on how to solve the problem rather than knowledge of the solution itself, this approach insulates against errors in the input. Errors and suboptimalities may slow down learning, but will tend not to affect solution quality. Further, using human input for scaffolding rather than direct answers focuses on higher level knowledge which tends to make more efficient use of users' time. I explore extraction of two different types of scaffolding, hierarchical decomposition and feature selection, covered in Sections 4.1 and 4.2, respectively. Although not every MDP will exhibit useful decompositions or benefit from feature selection, we find that when they do, **the scaffolding methods provide remarkable improvements to speed while insulating against errors**. This last benefit particularly highlights the potential of the scaffolding approach as it enables our methods to produce solutions which outperform end-users,

In Section 4.3, I move to issue (2). To do so, I turn to a paradigm known as Socially-Guided Machine Learning (SGML) [131]. This paradigm is motivated by the observation that while end-users are typically not experts in machine learning and have never taught a machine, they are experts in social teaching (*e.g.,* tutelage, imitation). Thus, to make teaching natural for end-users, one should tap into these social teaching skills. In particular, I will explore the role that interactivity plays in solicitation. Interactivity is known to play an important role in human learning [8, 7] and may be similarly helpful in an SGML setting. Through our user study, we will find that **interactivity is crucial to improving both learner performance and the teaching environment**. Interactivity helps by (1)

providing transparency of the learner, and (2) as a feedback mechanism, enabling end-users to adjust their teaching strategy.

4.1 LfD for Hierarchical Decomposition

Using human input for scaffolding translates to extracting hints, tips, and other supportive knowledge on *how* to solve the decision problem from human input. This indirect method of leveraging human input has the potential to significantly speedup learning while insulating against any errors in the input. In this section, we explore the use of human demonstrations to discover subtask decompositions and obtain speedup. In other words, we will use human input as a source of knowledge on the internal structure of the MDP.

Subtask decompositions break up an MDP based on identifying and factoring out subproblems that are repeatedly solved. It offers several advantages:

- Subtask decomposition breaks a problem up into several smaller subproblems. This provides a source of potential speedup since it is often faster to solve several smaller problems than one large one.
- Subtask decomposition enables us to avoid solving a subproblem multiple times. The ability to reuse (sub)solutions offers potential performance improvements.
- Decomposed subproblems and their solutions provide transfer learning opportunities. Although this will not necessarily improve scalability for the current problem, it does for future, similar problems.
- Subproblems can often take advantage of state and action abstractions not available to the global problem by taking advantage of its reduced scope. Abstraction can be a major source of performance gains as seen in prior work such as [34, 56].

On the other hand, finding good decompositions is tricky. The set of possible subproblems is exponential in the size of the state space, and it may be difficult to predict the speedup effects of factoring out any particular subproblem. One must also balance the cost of finding and solving subproblems with their benefits. Despite the difficulty, the problem has drawn significant interest.

While many methods for finding problem decompositions have been explored, our use of human demonstrations to discover subtask decompositions and obtain speedup is novel. Most previous work have tended to either assume too much or too little from their users. Early works in decomposition [34, 121, 98], for example, simply assumed that subtask decompositions are known *a priori*. End-users, however, typically have little understanding of the field. It would be unrealistic to expect them to provide such explicit decompositions.

Other methods have attempted a completely automated approach to finding decomposition. One approach is based on analysis of state features. This line of work obtains performance improvements primarily by leveraging abstractions that may be available in subproblems. For example, [62] assumes a dynamic Bayesian network (DBN) transition model and uses it to generate a causal graph depicting how state variables influence each other. It uses this causal graph to produce subtasks for changing the values of various state variables. Solving subtasks is fast because significant state abstraction is typically available. Hengst took a similar approach in [56], which also focuses on state features but uses their rate of change as a heuristic for finding decompositions. We note that without some additional information, these must be task independent methods of subtask decomposition. As such, the subtasks they find may not be ones that are frequently used, or even useful at all for the particular task at hand.

A different automated approach addresses frequency of reuse and focuses on state clustering or bottleneck detection. The intuition behind this line of work is that bottlenecks and inter-cluster connections tend to be frequently solved subproblems for a wide variety of general tasks. Examples of this work include those focused on single tasks such as [85, 84], as well as multi-tasks approaches such as [102]. In some examples along this line of work [85], bottleneck discovery is done *wrt.* the target task. While addressing subtask relevance and frequency, these works typically do not offer any abstraction opportunities — the main source of performance gains.

Our approach of leveraging human demonstrations to find decompositions strikes a middle ground. We ask additional information from our users, but of a form easy to obtain.

The closest relevant work is research performed by Mehta et al. in inducing MAXQ hierarchies [88]. Like [62], the authors in [88] assume a DBN model and use causal analysis, but they also take advantage of a single example trajectory to guide the subtask discovery. This allows them to induce a MAXQ hierarchy for the intended task. Unfortunately, the authors only evaluated their system on synthetic data and never with end-users so the final effectiveness is unknown. Further, their work sees the MAXQ hierarchy itself as the end as opposed to a means to obtain speedup. As a result, they do not weigh subtasks based on frequency and size, nor do they perform any estimation to see if a proposed decomposition will lead to performance gains. We address all these aspects as they contribute to potential performance improvement. Specifically, we:

- Look for subtask decompositions that provide abstraction.
- Focus on subtasks that are relevant for the intended task.
- Give greater weight to those subtasks that are used frequently.
- Estimate the benefit of performing any particular decomposition so that only those that would offer speedup are performed.

We will see that our focus allows us to discover subtask decompositions which provide speedups of over an magnitude. Importantly, this indirect method of using demonstrations enables us to obtain such speedups while providing robustness to any errors and suboptimalities within the demonstrations.

In the following discussion, we begin by providing some background and introducing some necessary notation. We then describe our approach and algorithm, Oplearn. Oplearn estimates the usefulness of proposed decomposition and only performs those deemed to be beneficial. Oplearn also computes the transition and reward model of decomposed subtasks so that their solutions may be injected directly into the original problem MDP as an additional action, enabling fast model based solutions. We will conclude this section with some analysis and empirical evaluations including a user study to measure Oplearn's effectiveness with actual end-users.

4.1.1 Notation

Subtask decomposition will require that we handle temporally extended actions. To do so we will use a Semi Markov Decision Process (SMDP) model. A SMDP $M = (S, A, T, R, \gamma)$ is defined similarly to an MDP and consists of a set of states S, a set of actions A, the transition function T, reward function R and discount factor γ. However, in a SMDP, the transition function $T(s, a)$ defines a joint probability distribution over the next state and the amount of time required to reach that next state. Thus, $T(s, a)(s') = \sum_{t=1}^{\infty} Pr(s', t|s, a)\gamma^t$ describes the likelihood of ending in state s' upon taking action a in state s over all possible durations t, appropriately discounted. Similarly, the reward function $R(s, a)$ specifies the expected, discounted reward accumulated over the duration of taking action a in state s. To simplify discussion, we will assume, without loss of generality, that rewards are negative.

Problem decomposition for (S)MDPs has been treated under several frameworks [34, 98]. In this paper we adopt the options framework [121]. An option (I, π, β) consists of three components. The initiation set, $I \subseteq S$, indicates the set of states where the option is available. The policy, $\pi : S \rightarrow A$, dictates the actions to be followed while the option is active. Finally, $\beta : S \rightarrow [0, 1]$ denotes the probability of terminating in any particular state.

We will assume states are represented by a set of features $f_1 \ldots f_n$, and thus $S = f_1 \times f_2 \times \ldots \times f_n$. A state abstraction $F \subseteq \{1 \ldots n\}$ is a set of indices. $S[F]$ refers to the state subspace induced by the cross product of the features whose indices are in F and is called the abstract state space of F.

Given state abstraction $\tilde{F}$ and F where $\tilde{F} \subseteq F$, we call a function $g : S[F] \rightarrow S[\tilde{F}]$ the *down projection* function. The function $h : S[\tilde{F}] \rightarrow 2^{S[F]}$ is the inverse function mapping a state from the space of $\tilde{F}$ to a corresponding set of states in $S[F]$ and is called the *up projection* function. Sometimes we want an up projection with respect to some state $s \in S[F]$. We will overload the function name h and define this up projection as $h(\tilde{s}; s) = x$ where x_i is $\tilde{s}_i$ if $i \in \tilde{F}$ and s_i otherwise. We call this the up projection of $\tilde{s}$ with context s.

We define a subproblem (M, F, A, ω) as a four tuple consisting of a base SMDP M, state abstraction F, action set A, and a goal $\omega \in S[F]$. F and A must be subsets of the feature space and action space of M. A subproblem induces an abstract SMDP via F and

A, which, when solved, yields a solution to the subproblem. We use the term "problem" loosely to refer to both SMDPs and subproblems.

A trajectory T of length k is a sequence of steps $t_1, ..., t_k$. Each step t_i is a 5-tuple (s, a, s', d, r) where s is the state before the action is taken, a is the action taken, s' is the resulting next state, d is the duration of the action and r is the (discounted) reward received over the duration of the step. An optimal trajectory is one that follows an optimal policy.

We will make use of the Taxi problem, a domain commonly seen in the literature, as a running example. Briefly, the Taxi domain is one in which an agent is a taxi whose objective is to move a passenger from a starting location to a desired destination. The world is a 5×5 grid. There are 4 pickup/dropoff locations each residing in one of the four corners of the grid: `NW`, `NE`, `SW`, and `SE`. The state space is composed of three state features: `taxiLocation`, `dropoffLocation`, and `passengerLocation`. The actions are *North*, *South*, *East*, *West*, *Pickup* and *Dropoff*. The MDP terminates when `passengerLocation` equals `dropoffLocation`. Reward is uniformly -1 except for invalid *Pickup* and *Dropoff* actions which yield -10. For details, see Section A.3.

4.1.2 The Oplearn Algorithm

We are interested in using demonstrations for discovering subproblems to speed up (S)MDP solvers and to provide future transfer opportunities. Ideally, we want to: (1) find a subproblem that can be solved quickly, (2) solve it, forming an option for performing the subproblem from its solution, and (3) insert the option into the original SMDP to lower its complexity.

The number of possible subproblems is prohibitively large so we would like to prune the set we consider. The reduced set should be significantly smaller but still contain most of the "good" subproblems. To ground our discussion, we have put together some characteristics of "good" subproblems. (1) **Size**: the subproblem should encapsulate a significant chunk of the overall problem. If this were not the case, learning the option would offer little overall savings. (2) **Frequency**: the more frequently a subproblem arises, the more savings the decomposition of the subproblem yields. (3) **Abstraction**: the greater the abstraction the faster we can solve the subproblem.

Our method of pruning the space of subproblems rests upon the following insight: the size and frequency characteristics of "good" subproblems reveal themselves in trajectories; namely, a subproblem of significant size and frequency leaves long, common action sequences that act as "signatures" which can be used to detect the subproblem. By finding these sequences, we can bias our search to those subproblems with significant size and frequency.

In order to judge the usefulness of a candidate subproblem, we need a way of estimating its benefit; that is, how much faster we can solve the overall problem if we factor out this particular piece, solve it, and then solve the rest of the problem. We perform this estimation by using the complexity of VI; however, this estimate requires knowing the maximum solution length (in terms of the number of steps) of the subproblem and the rest of the problem. Here, we again make use of the trajectories as they provide samples of the the subproblem and remaining problem length.

Algorithm 1 Oplearn

Require: SMDP M, trajectories T

let $subp$, T_{sub}, T_{rem}, $score$ = bestSubproblem (M, T)
if $score > 1$ **then**
  let M_{sub} = SMDP induced by $subp$
  let $subsol$ = recursively solve (M_{sub}, T_{sub})
  let o = create option ($subp$, $subsol$)
  let M_{rem} = add option o into M
  recursively solve (M_{rem}, T_{rem})
else
  let F = union state abstraction of actions in T
  let A = actions seen in T
  let M' = abstract SMDP(M, F, A)
  let sol = valueIteration (M')
end if
return sol

A high-level sketch[1] of our technique is presented in Algorithm 1. We require, as inputs, a set of trajectories and the SMDP providing the transition and reward models. We further assume that we have or can easily compute from the model, the set of features an action needs (features that affect or are affected by the action). For simplicity, we use value iteration (VI) as our baseline (S)MDP solver.

[1] Additional details can be found in our original publication [149].

Oplearn finds and solves subproblems in a greedy, depth-first manner. We only solve a (sub)problem directly when we estimate further decompositions to be detrimental.

We refer to the SMDP first passed to the algorithm as the **original** problem. In each (recursive) call of the algorithm, the SMDP in the argument is called the **base** problem. If a subproblem is identified and solved, the solution is made into an option which is added into the base SMDP. This produces a modified SMDP with reduced complexity. We refer to this as the **remaining** problem. We refer to trajectories similarly.

In our Taxi example, the original problem would be the full Taxi problem. The first subproblem discovered may be "pickup passenger". If so, our first step is to make a recursive call to solve this subproblem. In the recursive call, "pickup passenger" becomes the base problem. A subproblem discovered for it could be to "navigate" to a particular pickup location. Suppose there are no further decompositions for "navigate" so that it is abstracted and solved directly. The algorithm would then add the "navigate" option into the base SMDP. The remaining problem would be to figure out how to "pickup passenger" with the action set augmented by the "navigate" option.

4.1.3 Experiments

We first performed a series of experiments designed to explore the speedup of Oplearn under ideal conditions — conditions in which trajectories are assumed to be optimal, each corresponding to a successful episode from a random start state. We then study the robustness of Oplearn, examining the effect of varying numbers of trajectories and varying qualities of trajectories. Finally, we performed a user study to evaluate performance on demonstrations from end-users.

Note that in our results, we use number of operations (OPs) as a measure of speed instead of wall clock time. An operation is one expected value computation. This measure is similar to the number of backups but accounts for the size of the action set. OPs are machine independent, timing tool independent, and more reliable. Experiments measuring raw time show greater variation, but maintain the same trends.

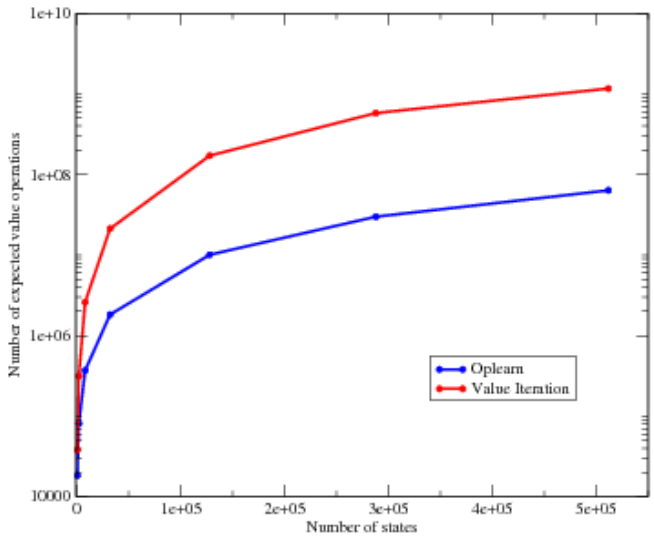

Figure 30: Oplearn and VI in terms of OPs

4.1.3.1 Speedup

To study what kinds of speedup Oplearn might offer and how that speedup is affected by the size of the domain, we created variants of the Taxi domain. VI is used as the baseline for comparison. To be fair, so that the trajectories do not offer Oplearn untoward advantage, when applying VI, we initialized its value table with value estimates from the trajectories. In these speedup experiments, Oplearn was given 10 optimal trajectories.

Figure 30 shows the speedup our technique yields over state spaces of increasing size. We generated different sized Taxi worlds by altering the size of the grid. Oplearn initially only yields about a 50 percent reduction in the number of operations. By the time the state space has reached 500k states, however, Oplearn reduces the number of operations by over an order of magnitude.

Examination of the options learned explain this behavior. Oplearn discovers options like "navigate to `NE` corner". As the size of the Taxi world increases, the frequency with which we see navigate options does not change, nor does the state abstraction of the subproblem. The length of the option, however, does change, and this results in increased savings on larger worlds.

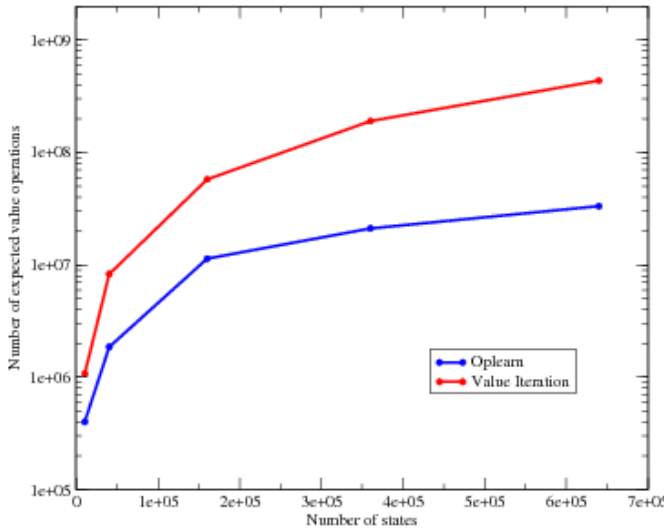

Figure 31: Oplearn and VI in two-person Taxi world

We are also interested in how speedup responds to additional state features. To measure this, we performed a similar set of experiments on a two-person Taxi world. In this variant, there are two passengers that need to be conveyed to their destinations. Thus, instead of three state variables, we have five: `taxiLocation`, `passengerA-location`, `passengerA-dropoff`, `passengerB-location`, and `passengerB-dropoff`,. As shown in Figure 31, we retain the trends seen previously; however, the slopes of the curves suggest greater savings. We believe this is because discovered options can ignore a larger percentage of the state features. For example, the navigate option needs only one-fifth of the state features, instead of one-third. Additionally, *Oplearn* does not produce the obvious decomposition of creating a single "pickup" option. Instead it learns separate options for A and B, allowing more state abstraction: when picking up A, all features *wrt.* B can be ignored and vice versa.

To ensure the results hold for non-deterministic worlds, we ran the same series of experiments on modified Taxi domains similar to the "fickle" version in [34]. In particular, actions only work 80 percent of the time. The rest of the time, they fail and leave the state unchanged. Results in the non-deterministic case maintain the behavior and trends displayed above although the savings are amplified. This is due to the ability of options

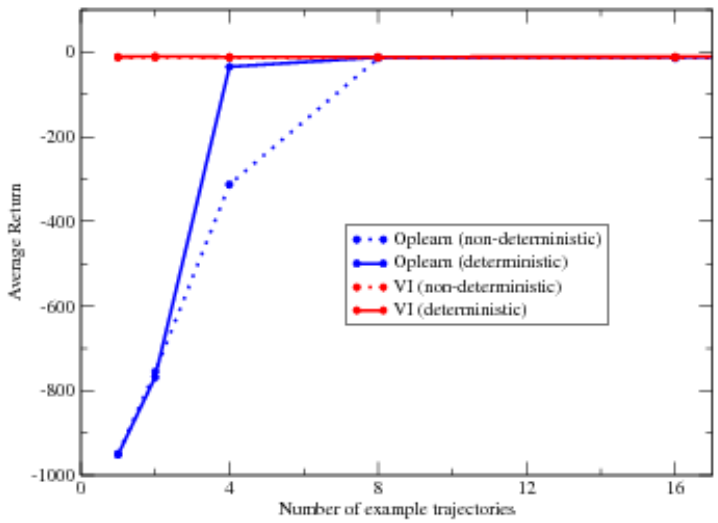

Figure 32: Optimality over the number of trajectories

to compartmentalize non-determinism. Due to non-determinism, navigating to the `NE` corner requires varying numbers of steps and accumulated rewards. Many iterations of VI are required before the value of a state will converge, and each of these iterations is over the full state space. By contrast, with Oplearn, the navigation task is factored out into a subproblem with an abstract state space consisting of just the taxi location state variable. Although it takes just as many iterations for the value to converge, each iteration is far cheaper because the abstract state space is much smaller. More to the point, the generated option will be deterministic, making the remaining problem much easier. The option "traps" the non-determinism inside itself.

4.1.3.2 Robustness

We ran two sets of experiments under tightly controlled settings to explore Oplearn's robustness . The first gauges the reaction to different numbers of trajectories, the second, to varying trajectory qualities.

Figure 32 shows Oplearn's response to the number of trajectories on the simple Taxi domain and its "fickle" variant. *Oplearn* performs near optimal as long as there are a sufficient number of trajectories. If there are too few trajectories, there may not be enough examples of how a subproblem is solved. Oplearn infers the abstract action set for a subproblem

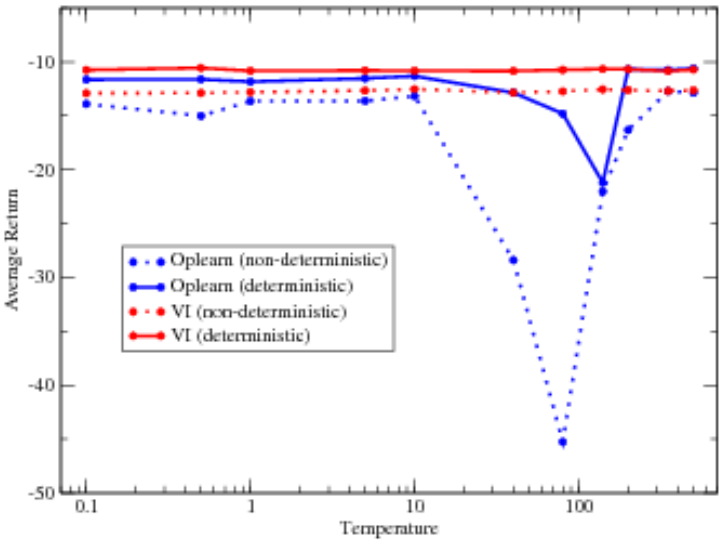

Figure 33: Optimality over increasing noise (temperature)

based on what actions it observes are used to solve the subproblem. As long as a variety of actions are seen, this is fine; too few, however, and the subproblem may not contain all the actions needed to solve it. Consider for example, the subproblem of navigating to the `NE` corner. If there are few example trajectories, it may just so happen that we only ever go to the `NE` corner from the `NW` corner. This would lead Oplearn to infer that only *East* is needed to solve the subproblem and produce suboptimal behavior. In practice, *Oplearn* is able to perform near optimally with just four trajectories in the deterministic setting and eight trajectories in the non-deterministic case.

To measure Oplearn's response to the quality of trajectories, we generated trajectories using softmax action selection. In particular, we choose action $a \in A$ with probability $\frac{exp(Q(a)/\tau)}{\sum_b exp(Q(b)/\tau)}$ where τ is a positive temperature parameter, and $Q(a)$ is the expected value of taking action a. High temperatures cause actions to be (nearly) equiprobable while low temperatures cause the action to be greedy. We simulate noisy trajectories by varying τ. Figure 33 shows the results. Oplearn was given 10 trajectories for this experiment.

Oplearn performs robustly. Oplearn maintains near optimality until temperatures of around 10. To give some intuition, at temperatures around 10, there is only a 30 percent

chance of choosing the optimal action, just twice as likely as random. The reason *Oplearn* is so robust is because it only uses the input trajectories to identify subproblems. Once found, the subproblems are solved independently. Thus, Oplearn routinely performs better than the example trajectories it learns from.

At temperatures beyond 10, optimality begins to fall because the trajectories have become so noisy that Oplearn begins to learn incomplete options. While this would not occur with sufficient trajectories, in this experiment, Oplearn must work with just 20 which is insufficient to avoid incomplete options.

One may be surprised that after temperatures of 200, optimality recovers. This is because by 200 the policy has sufficiently random that no options are found at all. This results in Oplearn reverting to baseline behavior. Oplearn suffers worse in the non-deterministic setting due to the additional randomness of the transitions.

4.1.3.3 User Study

To study Oplearn's performance on demonstrations from end-users, we performed a user study. We used ten participants solicited from the campus community. Their backgrounds range from Bachelor to PhD students. For the study, we used two more complex variations of the Taxi domain. The first is a larger, 20x20 version of Taxi with obstacles, see Figure 34. The second is a stochastic version of the first with many "wind" cells, see Figure 35.

In the study, we started by introducing the experiment and explaining that they would be asked to play two games several times so as to provide demonstrations for a computer learner. We then described the two games in detail, providing instructions on the rules and the goal of the game. We also provided a short demo of the important interface elements. Participants were given a few practice games to familiarize themselves with the system before starting the experiment. Once they were ready, they were asked to play the two taxi variations 15 times each. We logged demonstrated trajectories after each game of Taxi, for each participant in the study. At the end of the session, we gave a brief exit survey of demographic questions.

Figure 36 shows the results for the deterministic Taxi game using varying numbers of

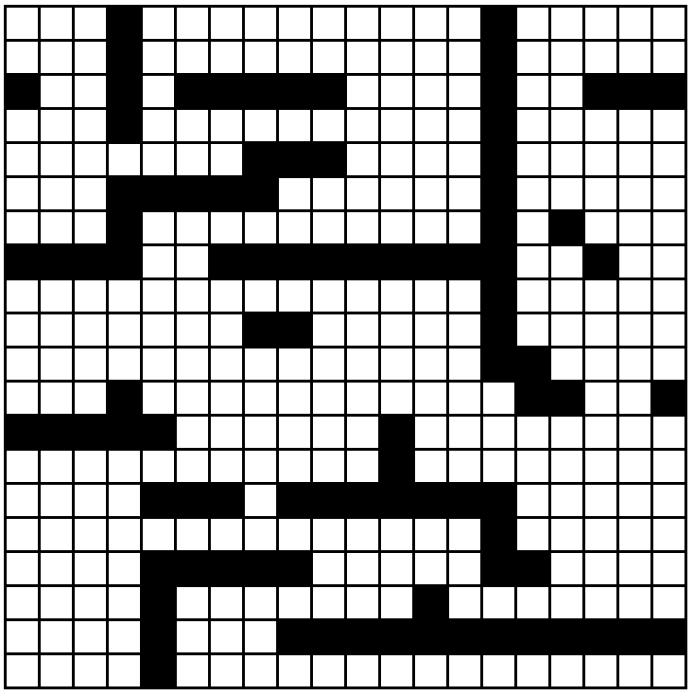

Figure 34: Deterministic Taxi domain used for the user study

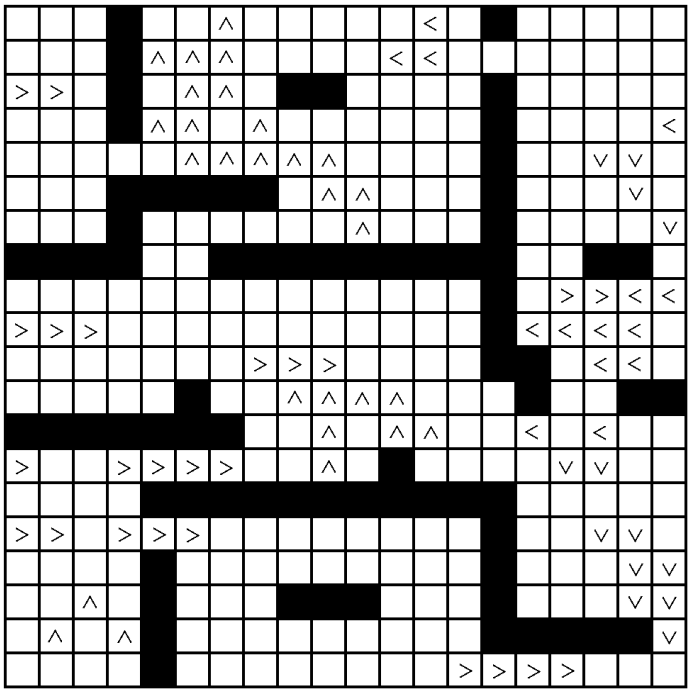

Figure 35: Non-deterministic Taxi domain used for the user study

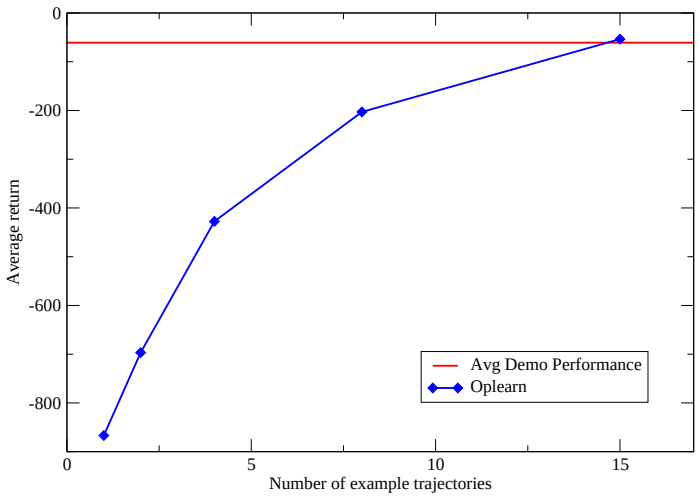

Figure 36: Optimality of Oplearn in the deterministic Taxi domain over the number of demonstrated trajectories used

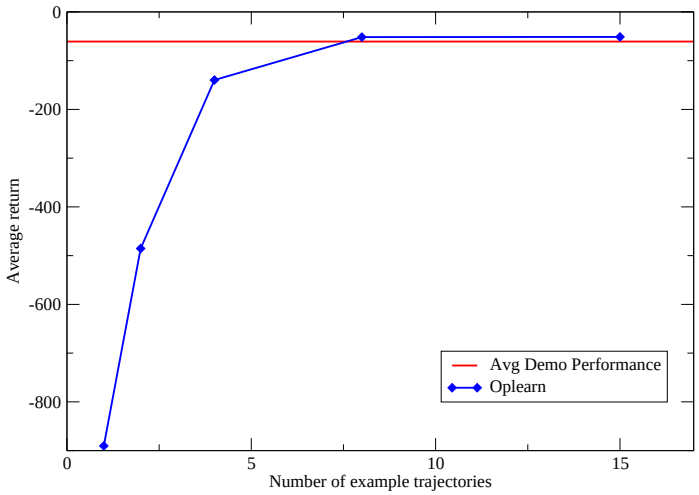

Figure 37: Optimality of Oplearn in the non-deterministic Taxi domain over the number of demonstrated trajectories used

demonstrated trajectories. Results are averaged over the ten participants. The flat, horizontal line shows the average demonstrated performance. As we can see, Oplearn quickly improves its performance after just a few example trajectories. By the time all 15 trajectories are used, Oplearn has achieved near-optimality, exceeding the participants' demonstrated performance. Figure 37 shows similar results for the stochastic Taxi game.

These results demonstrate that using demonstrations to find decompositions is an effective method of obtaining benefits from human input while hedging against errors in that input. They also demonstrate that this interaction scheme enables users to teach machines

to obtain better than teacher performance.

4.1.4 Discussion

Oplearn explores a particular form of using human input for scaffolding; it looks to extract subtask decompositions. Although its usefulness is limited to domains which contain subtask decompositions, it nevertheless provides a compelling proof of concept. By using demonstrations for decompositional information, it can obtain significant speedups while providing remarkable robustness to errors and suboptimalities within the demonstrations. Our user study further verifies its effectiveness with end-users, our target audience, and demonstrates the ability to obtain better than teacher performance.

While the speedups Oplearn is able to obtain is significant, it is not indicative of the speedup potential of decompositional approaches. Subtask decomposition works primarily by reducing the number of actions and the depth of the problem. However, Oplearn, being a proof of concept algorithm, uses VI as the base solver. VI only makes limited use of these advantages, its bottleneck lies in the size of the state space which does not change for the root task. To fully take advantage of subtask decompositions, sampling planners should be used. Sampling planners are independent of the size of the state space; their bottleneck lies in being exponential in the depth of the problem. With sampling planners, subtask decompositions can provide exponential speedups by reducing the depth of the problem, and by reducing the number of actions which lowers the branching factor of the planner.

4.2 LfD for Feature Selection

In this section, we continue our focus on leveraging human input for scaffolding. Specifically, we explore the use of human demonstrations to help identify relevant state variables, *i.e.,* for (state) feature selection.

In many MDPs the set of state features necessary to learn and represent a reasonable policy may be much smaller than the full set of features. Consider, for example, the sensor suite of a multipurpose robot which may contain everything from accelerometers to barometers. For a given task, only a subset of the sensors may be needed. Similarly, in hierarchical learning, subtasks typically only require a small set of the available features; even the root

task often only needs a small feature set given policies for accomplishing subtasks. Because MDPs are exponential in the number of state variables, reducing this number, *aka.* feature selection, stands to offer exponential gains in speedup.

Some prior works have considered feature selection for solving MDPs, however, they have been focused primarily on automated methods. Kolter, for example, has explored the use of L_1 regularization [71] which can be an effective means of removing unnecessary features. Others have explored L_2 regularization [40], and "forward-selection" style feature selection based on Bellman error analysis [65, 97]. While these methods can be very effective, they are limited in the features they can eliminate. They must keep sufficient features not only to represent the optimal (or near-optimal) value function. By contrast, we focus on features for representing the *policy*, and policies often require fewer features to represent than their corresponding value functions. More importantly, we need not represent the optimal policy. Our approach leverages an additional source of information, human demonstrations, which allows us to eliminate more features by focusing on just those features necessary to represent the specific policy demonstrated. These differences mean we have the opportunity to eliminate more features than possible with prior works.

We call our method, Abstraction from Demonstration or AfD. We will see that this form of scaffolding provides advantages similar to those seen in the decompositional form of scaffolding. Specifically, it offers large computational benefits while providing robustness to errors.

In the rest of this section we will present AfD, our algorithm for using demonstrations to identify relevant features. We will then present our empirical results on two different domains. As an additional baseline, we will provide comparisons to the direct LfD method.

In the discussion to follow, we will assume states are represented as a finite set of n features $S = \{F_1 \times \cdots \times F_n\}$ such that a state $s \in S$ as an n-tuple $s = (f_1, f_2, \ldots, f_n)$. A human demonstration is represented as simply a sequence of state-action pairs.

4.2.1 Abstraction from Demonstration

We present our algorithm for AfD[2] in Figure 38. AfD is composed of two main steps. It first builds an abstract state space S^α, and then it solves the MDP in that abstract space to produce the final policy. S^α is built from just those features necessary to predict the actions taken in the demonstrations. This feature set is computed by the feature selection algorithm F. To solve the abstract MDP, AfD uses a modified version of Monte Carlo with Exploring Starts [120]. The choice of a non-bootstrapping method is intentional. AfD reduces the feature set to just those features needed to represent a particular policy, *i.e.,* it collapses states with the same action together. This is known as policy invariant state abstractions. These abstractions are sufficient to represent a policy, but not to learn it under bootstrapping. Monte Carlo methods however, being non-bootstrapping, are safe [115]. This enables AfD to be "sound": in the limit of infinite data, the worst case policy performance of AfD is the same as that of direct LfD [27].

We tested two different feature selection algorithms for AfD. The first, which we call C4.5-greedy, is a simple, greedy, backward selection algorithm. Starting with the full feature set, it iteratively removes features (one at a time) that have little impact on the prediction accuracy of the decision tree learner. Specifically, in each iteration, the feature whose absence affects accuracy the least is removed. If the best feature to drop affects accuracy by more than 2% with respect to the current feature set, we stop. We also stop if dropping a feature results in an accuracy drop greater than 10% with respect to the original feature set. These stopping parameters are not sensitive. In experiments we have tested parameter values of 1% and 5% with no significant difference in the feature set selected. Note that we use *relative accuracy* for these stopping criterion, *i.e.,* , the amount of accuracy gained with respect to the majority classifier.

The second approach, Cfs+voting, uses Correlation-based Feature Subset Selection (Cfs) [53]. Cfs searches for features with high individual predictive ability but low mutual redundancy. The Cfs algorithm is used separately on the demonstrations of each individual.

[2]For more details see our original publication [27].

Require: MDP $M = (S, A, P^a_{ss^\alpha}, R^a_s, \gamma)$, $S = \{F_1 \times \ldots \times F_n\}$, feature selector F, human demonstrations $H = \{\{(s_1, a_1), (s_2, a_2), \ldots\}, \ldots\}, s \in S, a \in A$.
chosenFeatures ← F(H)
$\pi \leftarrow$ arbitrary policy
Initialize all $Q(s^\alpha, a)$ to arbitrary values
Initialize all Rewards[(s^α, a)] to []
while π performance has not converged **do**
  visited ← []
  Start episode with random state-action, then follow π
  for all Step (state s, action a, reward r) **do**
    for all (s^α, a) in *visited* **do**
      EpisodeReward[(s^α, a)] += r
    end for
    $s^\alpha \leftarrow$ getFeatureSubset(s, chosenFeatures)
    if (s^α, a) not in *visited* **then**
      Add (s^α, a) to *visited*
      EpisodeReward[(s^α, a)] ← 0
    end if
  end for
  for all (s^α, a) in *visited* **do**
    Add EpisodeReward[(s^α, a)] to Rewards[(s^α, a)]
    Q(s^α, a)=average(Rewards[(s^α, a)])
  end for
  Update greedy policy π
end while

Figure 38: Generic AfD algorithm with $\gamma = 1$, the general case is a simple extension.

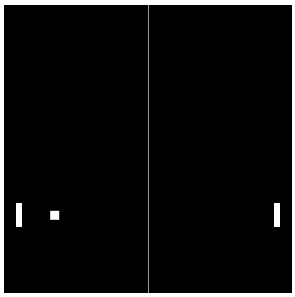

Figure 39: The Pong domain

A feature is chosen if it is included by at least half of the individuals.

4.2.2 Experiments

We ran experiments in two domains, *Pong* and *Frogger*. *Pong* is the simpler of the two domains where we focus on validating the algorithm with use of (near) optimal demonstrations. *Frogger* is the more complex domain where we focus on exploring the limits of our method and uses realistic input. In *Frogger*, we performed a user study, collecting demonstrations from the participants to ensure our approach is effective for our target audience. For each domain, we compare the performance of AfD to (1) our baseline solver, Sarsa(λ), (2) the demonstrations themselves, and as an additional baseline (3) the direct LfD solution using the C4.5 decision tree learner.

4.2.2.1 Pong

Pong is a small domain modeling a simple form of tennis where two paddles move to keep a ball in play, see Figure 39. The agent controls one paddle while the other is controlled by a fixed policy. In our experiment, the fixed policy moves the paddle in the direction that best matches the ball's Y position when the ball is approaching, and randomly otherwise. There are five state features in Pong: `paddle-Y`, `ball-X`, `ball-Y`, `ball-angle`, and `opponent-Y`. There are two possible actions: `Up` and `Down`. Reward is 0 except when successfully returning a ball, yielding +10. The game terminates when a player loses, or after a maximum of 400 steps indicating a draw. The optimal policy return is 60. Additional domain details can be found in Section A.4.

Table 2 shows our results. Near-optimal demonstrations are provided by the authors.

Table 2: Performance comparison of AfD on Pong.

Player	Average Return	Episodes
Sarsa	60.0	2554
AfD - C4.5-greedy	60.0	59
Human Demonstrations (authors)	56.5	—
Direct LfD	15.7	—

As we can see from the results, AfD offers over an order of magnitude speedup over Sarsa. Importantly, AfD does so while still producing an optimal policy. AfD's gains are made possible by the smaller abstract state space it is able to construct by removal of the unnecessary feature, `opponent-Y`. In other words, AfD was able to extract supportive knowledge from the demonstrations in the form of attention direction, and leverage this knowledge to solve the MDP 40x faster. Most importantly, this indirect approach enabled AfD to outperform the human demonstrations.

We also compared the direct LfD method to AfD where we observed a striking difference. AfD uses demonstrations as a source of supportive knowledge for speedup, but nevertheless needs to solve the MDP. As a result, AfD required more computation than the direct LfD method. However, this extra computation is compensated in two ways. First, as previously observed, AfD can obtain higher than teacher performance. Second, given identical training demonstrations, AfD yielded policies that performed five times better than that produced by direct LfD. This implies that AfD's indirect use of demonstrations for scaffolding makes more efficient use of human input than LfD's direct imitation approach.

4.2.2.2 Frogger

Frogger (see Figure 40), is a version of the classic video game by the same name. In the game, the protagonist frog, must move from the bottom row to the top row without falling in the water and without being run over.

In our Frogger domain, the screen is divided in a grid, and the state features are the contents of each cell relative to the current position of the frog. For example, the feature `3u2l` is the cell three rows up and two columns to the left of the current frog position, and the feature `X1r` is the cell just to the right of the frog. Possible cell values are `empty`, if the cell falls out of the screen; `good`, if the cell is safe; and `water`, `carR`, or `carL` for cells

Figure 40: The Frogger domain

containing water, or a car moving to the right or left. A total of 306 features are needed to cover all possible cells on the screen for all possible frog positions. Frogger has 5 possible actions: `Up`, `Down`, `Left`, `Right`, and `Wait`. Rewards are $r = 100$ for reaching the goal, $r = -100$ for death, and $r = -1$ for any other action. Additional domain details can be found in Section A.5.

In Frogger, demonstrations are not provided by the authors. Rather, we performed a user study and recruited human subjects to provide the demonstrations. This allows us to evaluate whether our strategy is effective for end-users.

We solicited 14 participants to provide demonstrations. In the study, participants were first read instructions describing the rules and goal of the game. They were then told they would be providing demonstrations for a computer learner. Each participant was given a few minutes to familiarize themselves with the game, after which they were asked to provide the demonstrations by playing the game for ten minutes. We logged the demonstrated trajectories after each game, for each participant in the study. At the end of the session, we gave a brief exit survey of demographic questions.

Demonstrations were preprocessed before use in AfD. Frogger is a real-time game and pauses are interpreted as `Wait` actions. However, most pauses are not intentional but are instead artifacts of unrelated events such as the user moving in their chair, or taking break. To remove these artifacts, we filtered out sequences of redundant samples. We also filtered out unsuccessful demonstrations. In our study, users provided an average of 33 demonstrations ($\sigma = 9.3$) totaling an average of 1230 state-action samples ($\sigma = 273.6$).

We present two sets of results. The first uses the aggregated samples of all users, 464

demonstrations (17221 samples) in total. The second focuses on just the demonstrations of the best player yielding 24 demonstrations (1252 samples).

Table 3 shows our aggregate results. Sarsa is not shown here because it could not complete. In our experiments, Sarsa had consumed 19GB of memory when it was killed. At that point, it had taken almost 1.7 million steps and the success rate was still below 0.2%. Two variants of AfD using different feature selection methods are studied. Both variants perform near identically suggesting some flexibility in the feature selection method used.

Results in Frogger confirm our previous results in Pong. First, AfD provides dramatic speedups over Sarsa. While Sarsa could not complete, having taken 1.7 million steps when it was killed, AfD converged to a near-optimal policy after roughly 100 thousand steps. Second, AfD is able to produce better than teacher performance, and yield almost perfect policies. Finally, AfD significantly outperforms direct LfD. This advantage is even more apparent when we look at our second set of results (see Table 4) from just the demonstrations of the best player. Even with just 7% of the demonstrations, AfD maintains the majority of its performance. By contrast, direct LfD suffers significantly. Comparing both tables, we can appreciate that AfD is much more sample efficient than LfD, performing better with 20x fewer demonstrations.

The superior performance of AfD reflects the fact that AfD does not learn directly from demonstrations but actually solves the underlying MDP. It obtains the best policy that can be expressed in the reduced feature space. In Frogger, AfD selects 9 to 12 features from the original 306, including key features such as the cells on both sides of the frog, and the three closest cells in the row immediately above the frog. Through the indirect use of demonstrations for scaffolding, AfD can effectively combine the ability of humans to discern relevant features with the optimization capabilities of the machine.

Table 3: Frogger domain, all demos (17221 samples).

Player	Success rate
Sarsa	N/A
AfD - Cfs+voting	97.0%
AfD - C4.5-greedy	97.4%
Human	31%-55%
Direct LfD	43.9%

Table 4: Frogger domain, best player demos (1252 samples). Note that Cfs+voting is not shown here because it is designed to work with many sets of demonstrations, not just one.

Player	Success rate
Sarsa	N/A
AfD - C4.5-greedy	88.3%
Human	55%
Direct LfD	17.1%

4.2.3 Discussion

AfD explores another form of using human input for scaffolding, it looks to demonstrations for feature selection. As empirical results show, it has advantages similar to Oplearn[3]: large speedups and a robustness to error that enables better than teacher performance. Additional comparisons to the direct LfD approach also demonstrates superior sample efficiency. However, subtle differences exist. Unlike Oplearn, AfD can provide exponential speedup to state-space limited algorithms because it removes state features which exponentially reduces the effective size of the state space. On the other hand, AfD only considers the problem globally. Unlike Oplearn, it cannot, for example, take advantage of more aggressive state abstractions available at the subtask level.

Finally, we note that the forms of scaffolding explored so far in both AfD and Oplearn provides robustness to error, not immunity. In both cases, it is possible, with cleverly designed demonstrations, to force these approaches into suboptimal solutions. However, as we will see in Chapter 5, with even more careful forms of scaffolding, it is possible to provide complete insulation from errors.

[3]See Section 4.1 for details.

4.3 Interactivity in Solicitation Environments

I now move to the second issue in leveraging human help, how to solicit input. To do so, I turn to a paradigm known as Socially Guided Machine Learning (SGML) [131]. SGML explores ways in which machine learning can be designed to more fully take advantage of natural human interaction and tutelage. It considers how solicitation should be performed and asks questions like "how do people want to teach agents?".

Inspiration for SGML comes (in part) from Situated Learning Theory, a field of study that looks at the social world of children and how it contributes to their development. In a situated learning interaction, the teaching and learning processes are tightly coupled. A good instructor maintains a mental model of the learner (*e.g.,* what is understood, what remains unknown) in order to dynamically adjusts their support based on the learner's demonstrated skill level. The learner, in turn, helps the instructor by making their learning process transparent through communicative acts, and by demonstrating their current knowledge and mastery of the task [8, 7]. Overall, this suggests that for machine learners to be successful, one must have a tightly coupled interaction in which the learner and instructor cooperate to simplify the task for each other.

In this section I address the impact of *interactivity*, the impact that a tightly versus loosely coupled interaction has on the learning process. Specifically, I will present results in which we compared two teaching paradigms: interactive LfD and batch LfD. In interactive LfD, the teacher provides a series of demonstrations interspersed with interactions with the learning agent, whereas in batch LfD, there are no interactions. Unlike sophisticated approaches to interactive LfD based on active learning principles [23, 26], we use a relatively simple framework. Our interactivity is one way. Human teachers see and evaluate agent performance but the agent has no direct communication channel to the teacher. For example, the agent cannot ask for specific demonstrations. Similarly, our LfD algorithm, a variant of off-policy Q-Learning, is also relatively simple. This simplicity reflects our focus on the effect of interactivity on learning. A complex interaction framework or LfD algorithm could conflate matters.

Our experimental testbed is a game platform where players teach a computer agent to

play Pac-Man. We find that:

- **Interactivity improves learning performance**. Specifically, we find that (1) interactivity improves player ability to evaluate the learner as well as the effects of their demonstrations, (2) players changed their teaching strategies based on these evaluations, and (3) adapted strategies resulted in faster learning.
- **Interactivity improves teacher experience**. Interactivity makes players feel more engaged. Specifically, they (1) felt like they made a difference and (2) felt like an active participant. Interactivity may also encourage players to participate longer.

The first empirical result is particularly interesting because it implies that the improved transparency provided by interactivity has a dual channel effect. First, transparency *prior* to player guidance has been shown to improve learning performance [131, 130] — before a player gives input, it is helpful to know the state of the learner. Second, the extended transparency provided by interactivity gives insight into the learner *after* player guidance, enabling players to see the effects their inputs had. This forms a feedback mechanism allowing players to adapt their teaching strategies based on their effectiveness.

In the discussion to follow, I will begin by detailing the elements of our study and then present its experimental design and procedure. I will then present the results and consider their implications.

4.3.1 Platform

4.3.1.1 Domain (Pac-Man)

We used the Pac-Man game as the platform of our user study. Pac-Man is a classic arcade game from 1980. The original version of Pac-Man (see Figure 41) is a single-player game where a human player controls the Pac-Man character around a maze. Pac-Man must avoid four ghost characters, Blinky, Pinky, Inky and Clyde, while eating dots initially distributed throughout the maze. When all dots are eaten, Pac-Man will be taken to the next level. If Pac-Man is caught by a ghost, he loses a life. When all lives (usually three) have been lost, the game ends. Near corners of the maze, there are large, flashing dots known as power

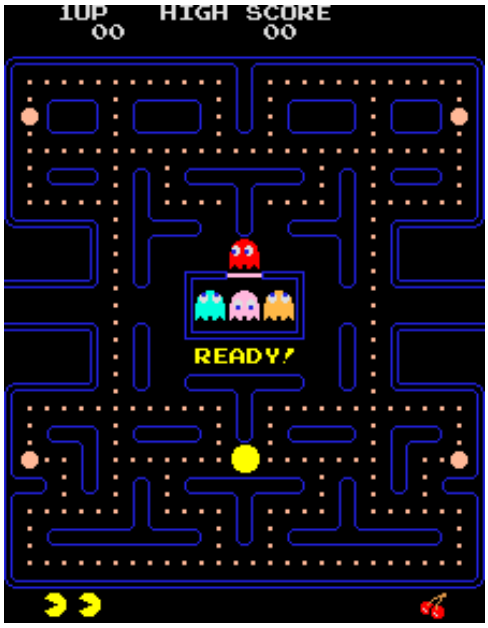

Figure 41: Screenshot of the original arcade game, Pac-Man

dots. Eating a power dot gives Pac-Man the temporary ability to eat the ghosts. When a ghost is eaten, it returns to the ghost spawn location ("ghost jail"). In the typical version of the game, normal dots are worth 10 points, power dots are worth 50 points, and ghosts are worth 200, 400, 800, and 1600 points for the first, second, third and fourth ghosts.

Our platform uses a scaled down variant of Pac-Man. This enables our learning algorithm to learn quickly enough to be used in a real-time fashion, allowing participants to immediately see the results of their training demonstrations (in the interactive version). In our version of Pac-Man, the maze is smaller: 7 x 8. There are also fewer dots: eight normal dots and one power dot. Only one ghost, Blinky, roams the maze. Blinky is the chaser ghost, it always chases Pac-Man using Manhattan distance as its heuristic. In our scaled down variant of Pac-Man, there is only a single level. The game ends when you beat the level. Scoring is done as follows: 1000 bonus points are awarded for clearing a level (when all dots are eaten), 10 points are awarded for eating a dot (both normal dots and power dots), 100 points are awarded for eating the ghost, and 1000 points are deducted if Pac-Man is caught by the ghost. Maximum score is 1190 (1000 + 90 + 100).

4.3.1.2 Learning Algorithm (Q-Learning)

In our domain, the state is represented by a vector containing the position of Pac-Man, the position of the ghost, nine binary variables denoting the existence of the nine dots, a jail counter denoting the remaining time the ghost must remain in its jail, and a power-mode counter representing the time remaining in power-mode. The actions are `NORTH`, `SOUTH`, `EAST`, and `WEST`. The reward function corresponds to the points. That is to say, eating a dot results in +10 reward, eating a ghost is +100, *etc.*

We used Q-Learning, for our learning agent. Q-Learning [140], is widely used in reinforcement learning [130, 61, 116] and has had many positive results in playing stochastic games [77], elevator control [28] and robotics [110]. Q-Learning produces an action-value function which maps every state-action pair to the expected utility of taking the action in the state and following the greedy policy thereafter. The greedy policy of a Q function is one that simply chooses the action with the highest Q-value for any state.

Q-Learning is an off-policy learning algorithm, meaning it can learn from example trajectories that differ from its policy. To learn from demonstrations, we take advantage of this ability and simply perform Q-Learning along the demonstrated trajectory.

We implemented a variant of the Q-Learning algorithm. For human trajectories, we used a simple replay mechanism, BTD [25], to update the Q-values. This magnifies the effect of human demonstrations and speeds up the learning process. Simple replay is not performed for self-play trajectories as they are likely less optimal and contain less accurate Q estimates.

In our implementation, Q-values are randomly initialized. We set learning rate α to 0.8, discount factor γ to 0.99, and ϵ-greedy exploration parameter ϵ to 0.9, which decays with rate 0.95 per game. Note that in the interactive learning case, ϵ is set to zero once interaction begins (after game 45). This ensures the agent performance players observe is representative of the underlying learned Q-function and not due to a random exploratory action being chosen.

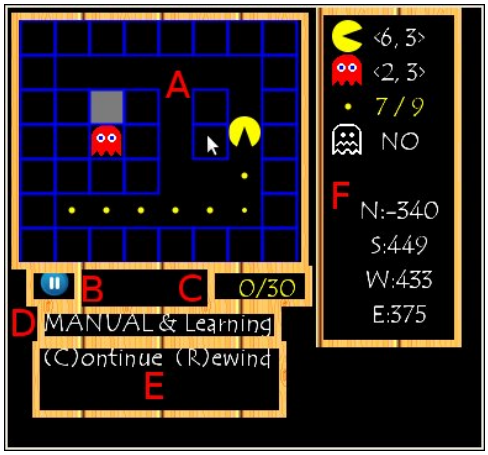

Figure 42: Screenshot of our interface

4.3.1.3 Interface

We set up an interface for soliciting demonstrations. The interface (see Figure 42), has several components detailed in the list below and tagged in the figure as A-F. The first is the Pac-Man board (A), the second contains the demonstration controls (B, C, D, E). The right hand side (F) contains debugging information and can be ignored.

A: Pac-Man board

B: Pause/Unpause button and indicator

C: Shows number of human demonstrations stored

D: Shows the current play mode.

"MANUAL & Learning" means the game is in player's control and the agent is updating Q-values.

"AUTO & Learning" means agent is making moves while updating Q-values.

"MANUAL & Playing" means player is playing the game and the agent is not learning.

E: Available control keys. 'C'ontinue is equivalent to the unpause button. Pressing 'R' rewinds the state of the game by one step. It can be repeatedly used to rewind the game to the beginning. Other control keys (not shown) are: 'P' for pausing the game, 'S' for starting a new game.

F: Debugging information. Ignored in the experiments.

4.3.2 Experimental Design

We ran a user study with our platform to see how interactivity affects both agent learning and the teaching experience. We used a randomized, between groups study of 20 participants solicited from the campus community. Their backgrounds range from Bachelor to PhD students, as reported in an exit survey. Each participant was assigned to participate in one of the two groups for a total of 10 participants in each group:

- BATCH: This group gave demonstrations directly without feedback or interaction.
- INTERACTIVE: This group gave the second half of their demonstrations with interaction.

Batch learning mode: Players were asked to demonstrate 30 consecutive games. They were allowed to rewind and correct their trajectory if they wished (*e.g.,* to correct an unintentional mistake). They received no feedback on how the agent learned. After the 30 demonstrations, the agent played 60 additional games to learn on its own. Thus, the final policy is a result of learning from a total of 90 trajectories.

Interactive learning mode: This mode consists of two parts. In the first half, players were asked to demonstrate 15 games just as in batch mode. The agent was then allowed to learn on its own for another 30 games (this process was performed opaquely, with all animations showing agent actions turned off). After this initial learning period, the second half began: we restarted animations and allowed players to watch the agent as it played and learned on its own, *i.e.,* they were able to watch the agent controlled Pac-Man move around on the board. For 45 more games, players were asked to watch agent play, and if they deemed necessary, provide corrective demonstrations. To provide a corrective demonstration, the player must pause the game (or wait until Pac-Man dies) and rewind play to an appropriate point from which to provide a demonstration of what Pac-Man should have done. In other words, the player plays Pac-Man to completion from the rewound point. Players were allowed a maximum of 15 corrective demonstrations. As with the batch mode, the

final policy is a result of learning from a total of 90 trajectories. However, unlike batch mode where 30 of the total 90 trajectories are from player demonstrations, the number of player demonstrations here may range anywhere from 15 to 30 depending on the number of corrective demonstrations the player provides.

In the study (for all groups), participants were first given instructions to read. The instructions describe the rules and goal of the game. It also introduces the player to their role as a teacher, to teach the computer to play Pac-Man. Depending on which group a participant is in, we then gave specific, group appropriate instructions outlining the study progression. Each participant was compensated $5 for participation in the study. To help participants understand the instructions, we provided a short demo of the important interface elements. Participants were then allowed to play a few practice games before starting the real experiment. When participants reported being ready, we started the experiment. At the end of the session, an exit survey (including demographic questions) and a brief interview were conducted. Questions were mainly about teaching strategies, perceived performance of agent's learning, and comments on the teaching environment.

We logged the learned policy after each game of Pac-Man for each participant in the study. We also maintained trajectory logs for each participant. It recorded state transitions, actions, rewards, and value-updates.

4.3.3 Results

We cover the results in two parts. In the first, we will focus on the impact of interactivity on learning. In the second, we will focus on its effect on the teaching experience.

To evaluate the effect of interactivity on learning, we compare logs of agent performance between interactive and batch groups. Figure 43 shows the averaged learning curves of the two groups. As the number of games (trajectories) available to the agent increases, so does its performance. Note that the composition of the games is different between the groups. The first thirty games of the batch group are human provided training games, the rest are games generated from self-play. By contrast, only the first fifteen games of the interactive group are human provided, the following thirty games are generated from self-play, and the

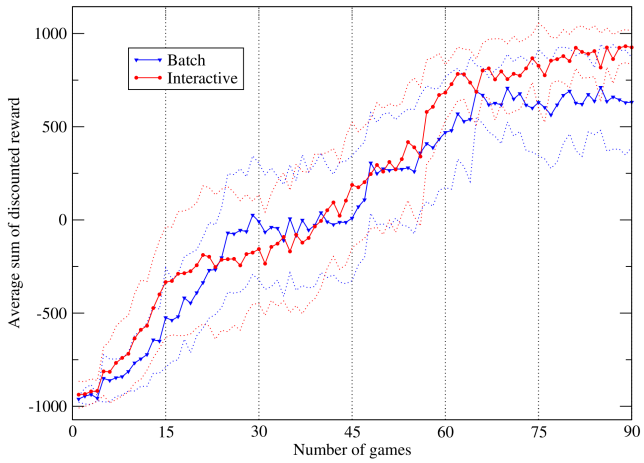

Figure 43: Averaged learning curve for batch and interactive. Dotted curves show the lower and upper 95 percent confidence interval band.

final games (from game 45 onwards) are a mix of human demonstrations and agent self-play.

We can see that initially, batch and interactive performances are comparable. The first time the interactive group statistically significantly outperforms the batch group occurs at game 63 (df=14.1, t=1.79, p=0.048). While not every game thereafter has a statistically significant difference, by the end the difference is clear: game 88 (df=11.0, t=1.96, p=0.037), game 89 (df=10.8, t=2.01, p=0.035), game 90 (df=11.45, t=2.24, p=0.023). The difference is even more evident when we consider that interactive participants only gave a total of 24 teaching demonstrations on average as compared with the 30 that batch participants gave (df=9, t=3.86, p=0.002). Players in the interactive group gave (on average) fewer demonstrations because the agents often began performing very well after just a few and did not need further help. We can also compare performance between the two groups in terms of win percentage. Under this intuitive metric, the difference is even more clear. The interactive group produced agents with an average win rate of 92%, close to the best possible within the experiment[4]. By contrast, the batch group has an average win rate of 77%. The difference is significant (df=11.5, t=2.41, p=0.017).

To explain why interactivity improves learning performance we hypothesize a dual channel effect based on transparency. Providing transparency *prior to* player guidance enables

[4]Achieving a perfect win rate is very difficult when there are not many games. Participants often do not get a chance to correct errors in the agent's policy simply because they are not encountered.

the player to determine if guidance is needed, and if so, helps them to match their guidance to the needs of the learner. This type of transparency is the first channel of effect and studied in previous works [131, 130], where it has been shown to provide a beneficial effect. The second type of transparency is that provided *after* player guidance. This type of transparency forms a feedback mechanism enabling players to see the result of their guidance so that they may adjust it accordingly. This second channel of effect can only be created through prolonged transparency such as that provided through interactivity.

To support our hypothesis, we first focus on the fundamental assertion that interactivity provides transparency. To do so, we will (A) show that players from the interactive group are better able to evaluate the learner. Once transparency is established, we will move to study its dual-channel effect. Since the first channel has been previously studied and its effects known, we will focus on demonstrating the second (feedback) channel of effect. Specifically, we will show that the feedback provided by extended transparency into learner performance (B) gives players a better perception of the effects of their actions, and (C) enables them to adapt their teaching strategy based on it. Finally, we will show that (D) adapted strategies result in improved learning.

4.3.3.1 A. Interactivity improves ability to evaluate the learner

In our exit interview, participants were asked to give an estimate of the performance of the final trained agent. Not all participants were willing to give an estimate, some reported that they were unsure and had no estimate to give. This occurred frequently in the batch group where half of the participants could not estimate agent behavior. By contrast, all participants in the interactive group were able to give estimates. This difference is statistically significant: df=1, $\chi^2 = 10$, p=0.0016.

Although interactive participants are better able to estimate learner performance, the estimates must also be accurate to be useful. In exit interview responses, all interactive participants estimated the final agent to perform "pretty well, though not perfect". This is borne out in empirical results. Actual performance of the final agent has a 95 percent confidence interval from a score of 836 to 1015, or alternatively, a win rate of 88 to 96

percent. This corresponds to player estimates, leading us to conclude that interactivity provides better evaluations of learner performance and thus provides greater transparency of the learner's state.

4.3.3.2 B. Interactivity improves perception of player effects

We hypothesized that interactivity enables players to better perceive the effects of their actions. To test this, we asked participants whether they could tell if the demonstrations they provided made a difference. In the batch group, only half of the participants replied positively. The other half indicated that they could not determine if their demonstrations made any difference due to a lack of information. By contrast, all participants in the interactive group replied positively. This difference is statistically significant: df=1, $\chi^2 = 10$, p=0.0016.

4.3.3.3 C. Participants adapted their teaching strategy based on feedback

In exit interviews, 70 percent of interactive participants said they changed their teaching strategy during the session. To verify this empirically, we compared their teaching strategy in the first fifteen games before they have a chance to see learner performance with the strategy used in the remaining teaching demonstrations. A "strategy" is modeled as a partial policy, $\pi : S \rightarrow A\,\text{distribution} \bigcup \perp$, mapping states to action distributions where $\perp$ indicates that the policy is not defined for the given state. We estimate players' teaching strategy from their demonstrations. This is best illustrated by an example. Suppose we see the following demonstration (trajectory) of state/action pairs: `(A, East); (B, East); (C, West); (B, North)` This would create the partial policy shown in Table 5.

Table 5: Example partial policy

State	**Action distribution (N/S/E/W)**
A	0% / 0% / 100% / 0%
B	50% / 0% / 50% / 0%
C	0% / 0% / 0% / 100%
all others	$\perp$

We compare two strategies or partial policies by computing the average difference between action distributions on those common states defined in both strategies. More formally the difference between two strategies is computed as $d(\pi_a, \pi_b) = \frac{1}{Z} \sum_{s \in \pi_a \bigcap \pi_b} L2E(\pi_a(s), \pi_b(s))$, where $L2E$ is the integrated square error [113] and Z is a normalization constant.

When we looked at the policy difference between the first fifteen and the later demonstrations, *i.e.,* the shift in teaching strategy, we saw a statistically significant difference for those participants who said they changed policies over those who said they did not: df=7.56, t=3.18, p=0.0065. In other words, participants who said they changed policies actually did.

To test whether the change was based (at least in part) on feedback provided by the system in the form of observed learner performance, we compared the amount of policy change of players with the five worst initial agents, with the amount of policy change of participants with the five best initial agents. By "initial" we mean the agent's performance after the first fifteen demonstrations, before players have a chance to interact with the learner.

A t-test shows that the players with poor initial agents changed their teaching strategy significantly more than those with good initial agents: df=6.70, t=5.97, p=0.00027. We also performed linear analysis of the data (see Figure 44) and found that relative initial performance of the agent is a good predictor of the amount of policy change (r^2=0.83). Relative performance is computed as $(score - minscore)/(maxscore - minscore)$.

These results show that players adapted their teaching strategies based on feedback provided by seeing how well their learner performed in response to their demonstrations. The worse the learner responded to their teaching strategy, the more dramatically the player would change the strategy.

4.3.3.4 D. Adapted strategy results in faster learning

To see whether adapted strategies resulted in faster learning, we first examine learning rates. We compare the learning rate of interactive participants that significantly adapted their strategy, with interactive participants that did not. Figure 45 compares the two learning curves.

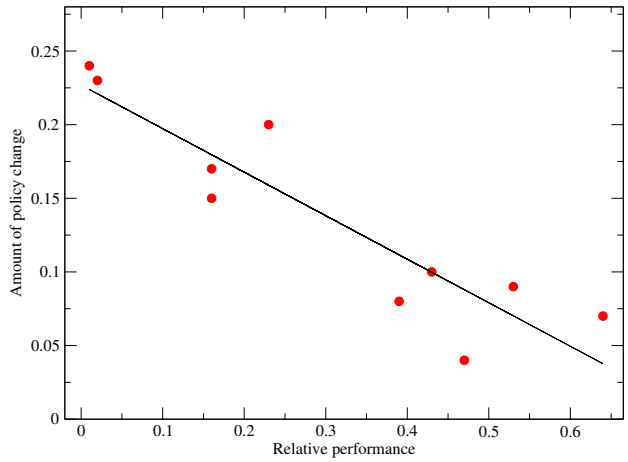

Figure 44: Amount of policy change as a function of relative initial performance.

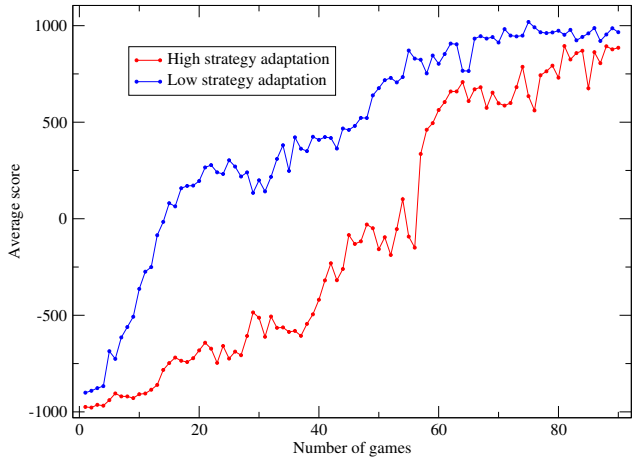

Figure 45: Averaged learning curve of interactive participants that significantly changed their teaching strategy and those that did not.

For interactive participants that significantly adapted their strategy, the most rapid improvement (highest learning rate) occurred around game 60 (mean=59.7, stderr=2.9) meaning it occurred with 95% confidence somewhere in the range of game [54.0, 65.4]. This range exactly corresponds with when the additional, adapted strategies began to make up a significant portion of the teaching demonstrations. The range corresponds to [5.8, 9.2] adapted games which proportionally is [27.9%, 38.0%] of training demonstrations. We can see the resulting rapid improvement in agent performance at this range in Figure 45. Clearly, adapted strategies accelerated the learning rate.

By contrast, for interactive participants that did not significantly adapt their policy, the

most rapid improvement came around game 21 (mu=20.6, stderr=7.4) meaning it occurred with 95% confidence somewhere in games [6.1, 35.1]. This range occurs within the first half of the interactive mode, somewhere during the initial 15 examples and subsequent self-learning. This aligns with typical learning curves in reinforcement learning literature where learning occurs at its fastest rate relatively early on, before slowing down and gradually leveling off.

This difference in the location of most rapid learning, between those who did and did not change strategies, is statistically significant: df=5.19, t=4.91, p=0.002.

A second way we can tell that adapted strategies resulted in faster learning is by examining when the performance difference between interactive and batch groups first becomes statistically significant (see Fig. 43). This occurred at game 63 (df=14.1, t=1.79, p=0.048), which again corresponds to when participants' adapted strategies start making up a significant portion of the teaching demonstrations.

Now we will consider the effect of interactivity on the teaching experience. We found the effect to be positive. In particular, it makes players feel more engaged and may encourage them to participate longer which is inherently beneficial to the machine learner.

4.3.3.5 Longer participation

Now that we have considered the impact of interactivity on learning, we will move to focus on the effect is has on the teaching experience.

We begin by comparing participation length. We asked participants how much longer they would perform the teaching task before they grew bored or tired of the task. 70 percent of batch participants reported they tired of the task within the study length, while 30 percent reported interest in continuing longer. Interactive participants, on the other hand, reported qualitatively different results. Many gave length estimates dependent on learner performance.

When batch participants describe how long they would perform the teaching task, all measured length by units of time (*e.g.,* minutes) or by number of games. In other words, when asked how long they would perform the task before tiring of it, they reported things

like “a little more, maybe 5 to 10 minutes” or “20 more games”.

By contrast, half of the interactive participants (5 of 10) reported variable lengths. Specifically, four reported they would remain interested until the agent performed satisfactorily, and the fifth reported interest until the agent stopped improving. Of the remaining half, three reported a length less than or equal to the study length, and the remaining two reported a length greater than the study length.

If we assume that it takes more than 30 games (the study length) to obtain good performance and that the learner can show steady improvement, then 70 percent of the interactive participants would perform the task longer than the study length. In our hypothetical scenario, this would lead to interactive participants being interested in spending statistically significantly more time on the task (df=1, $\chi^2 = 7.62$, p=0.0058). Unfortunately, our task was not sufficiently challenging. It does however, lead us to speculate that given a sufficiently challenging problem, and a learner that shows steady improvement, future experiments would be able to demonstrate interactivity to encourage longer participation.

4.3.3.6 More engagement

Participants in the interactive group felt more engaged than those in the batch group. We can already see this from previous results on participation length. In the interactive group, half of the participants described the length of time they would be willing to spend in terms of agent performance, meaning they took stock of the agent and its performance. By contrast, none of the batch participants did. This statistically significant difference (df=1, Chi-squared=10, p-value (two-tailed): 0.0016) suggests a higher level of engagement.

Responses to two other exit interview questions lead us to conclude the interactive group were more engaged. We asked participants whether they “felt like [they] made a difference” and whether they “felt like an active participant”. In the interactive group all participants responded positively to both questions. By contrast, in the batch group, only 50 percent and 60 percent of participants, respectively, answered positively to the questions. This difference is statistically significant for the difference question (df=1, $\chi^2 = 10$, p=0.0016) and also for the active participant question (df=1, $\chi^2 = 6.7$, p=0.0098).

4.3.4 Discussion

Our study of interactivity demonstrated its importance to both learner performance and teacher experience. The results are particularly interesting because we studied a passive form of interactivity in which the learner has no direct communication channel to the teacher, and so cannot, for example, ask for specific demonstrations. This implies that the learner is able to improve its own learning environment through transparency alone. We also gained a better understanding of how transparency is able to obtain these improvements; we saw that it has a dual-channel effect. First, transparency provides insight into the learner, enabling the teacher to tailor their guidance to the needs of the learner. However, extended transparency produces a second mode of action; it becomes a feedback mechanism giving teachers a better understanding of the effects of their guidance. This improved understanding enables teachers to adapt their teaching strategy based on its effectiveness, in an online fashion.

In the course of our study, we also learned that seeing learner improvement increases teacher engagement and that teacher engagement and interest wanes as learner performance becomes more satisfactory. This suggests that humans tend to be more interested in satisficing as opposed to optimal behavior. As a result, methods which leverage human input such as LfD, may be more suited for obtaining the initial jump in performance than for refining a policy to optimum. It also suggests that learning algorithms that can show steady, consistent improvements are more suitable for use with LfD when demonstrations are obtained from end-users.

The results of our study also has implications for the larger question of how to solicit human input. First, they imply that we should design solicitation interfaces to provide interaction and transparency to the learner. Second, they imply that we should focus on learners with useful intermediate results, *i.e.,* anytime algorithms. Finally, the results demonstrate the applicability of principles from how humans learn and teach from each other, to SGML, suggesting that this may be a fruitful avenue of further exploration.

4.4 Summary

In this chapter, we examined the problem of effectively leveraging human input. We pointed out that the most common form of human input, those from end-users, need to be carefully solicited because users typically have no experience teaching machines and the help they provide often includes errors. We then focused on two specific issues (1) how to insulate against potential errors in user inputs, and (2) how to solicit input. We explored these issues in a LfD context.

To explore the issue of insulating against errors, we took the general approach of treating user input as a source of scaffolding. To this end, we explored two different scaffolding methods of using demonstrations. The first used demonstrations to discover subtask decompositions, while the second used demonstrations to find relevant features. In both cases, we found that using demonstrations as a source of scaffolding was more effective than using them directly for answers. Although this approach is indirect, it enables us to learn from human inputs while being robust to user errors. Additionally, it makes more efficient use of user demonstrations. We saw that policies learned in the indirect fashion could not only outperform directly learned policies, but the users themselves as well. The price we pay for these benefits is additional computation. The indirect method requires that we solve the (albeit simplified) underlying MDP from its reward signal. By contrast, a direct LfD approach reduces the MDP into a supervised learning problem. We argue that the indirect method strikes a useful compromise. It provides enough computational savings to make the problem tractable, but otherwise places the bulk of the work on the machine rather than the user. Since CPU time is much cheaper than human time, it seems like a good trade-off.

To explore the second issue, how to solicit input, we turned to the paradigm of SGML. Motivated by the role interactivity plays in Situated Learning Theory, we studied the impact of interactivity on learning performance and teaching experience in a SGML setting. In our study in which human subjects were asked to teach a computer to play Pac-Man, we found that interactivity improved both learning performance and player experience. In particular, we found that interactivity improved the ability of the player to evaluate learner performance and the effects of their guidance, that players adapted their teaching strategies based on

these evaluations, and that the resulting adapted strategies resulted in faster learning. With respect to player experience, we found that interactivity made players feel more like they made a difference and more like an active participant. We also saw that learner improvement increases teacher engagement and that teacher engagement and interest wanes as learner performance becomes more satisfactory. These results suggest that human help is best suited for obtaining the initial jump in learning performance. They also suggest that learning algorithms whose progress is easy to convey, *e.g.,* those with intermediate results that can show the algorithm making consistent, significant improvements, will be more effective at obtaining human help.

Finally, we note that demonstrations appear to be an effective means for obtaining supportive knowledge difficult to obtain otherwise. Asking users to select relevant features directly, for example, is problematic because users are generally not familiar with the technical definition of "feature" and many features may not have readily interpretable meanings. Demonstrations allow us to get at users' tacit knowledge. We also note that it appears to have a relatively low cost. In both the scaffolding methods we explored, the computational cost was dominated by solving the simplified MDP.

CHAPTER V

COMBINING HUMAN INPUT WITH APPROXIMATION

In Chapter 3, I explored the use of approximation in solving MDPs and developed a general solution method, EVA, which works by building successively expanding approximations of the value function. Importantly, I showed that EVA solves MDPs with guarantees of convergence and bounded error. I then explored the use of human input in Chapter 4. There, we saw the benefits of using human input as a source of scaffolding, particularly with regards to robustness to input errors. We also saw the importance of interactivity in the solicitation of human input, how it not only improved learner performance (*i.e.,* the machine's performance) but the teaching environment as well. In this chapter, I will bring these results together. Specifically, I will show how the approximation algorithm EVA can fruitfully take advantage of human input, and how this joint system can offer several orders of magnitude speedup over tabular methods while maintaining the theoretical properties of EVA.

Combining human input with EVA cannot be performed by directly applying the techniques we developed in Chapter 4 for leveraging human input. One problem with these previous techniques is that they focused on tabular methods. As a result, they looked to state abstractions as the primary source of speedup. While state abstractions can offer exponential reductions to the size of the space and yield exponential speedups for tabular methods, EVA is not a tabular method. EVA relies upon function approximation, where the size of the state space has little impact. Instead, what matters is the complexity of the function to be approximated, and specifically, how amenable that function is to the inductive bias of the approximator being used. A second problem of our previous techniques is that they focused around demonstrations. While demonstrations are rich in information, much of their benefits are lost when applied to EVA. EVA works through a series of expansion rounds in which each round leverages the results of previous expansions. It

is typically faster for the machine to solve any given expansion state than for the user to provide demonstrations for that state. For these and other reasons, direct application of the techniques we developed in Chapter 4 would be inappropriate, and yield little benefit.

To effectively make use of human input in conjunction with EVA, our previous results must be applied at a higher level. That is to say, the results should be taken as lessons to be applied around a new interaction scheme, one that matches the way EVA operates. To this end, we developed training regimens, an interaction scheme in which the user provides a series of increasingly difficult starting positions. Training regimens are like demonstrations, but without the solutions; they contain just a series of (starting) states. Applying lessons of scaffolding and interactivity, we developed two variations for using training regimens with EVA. The first variation provides more flexibility to the user, while the second is cruder but less time consuming. In empirical studies, we will see that both can provide over an order of magnitude speedup to EVA. We will also see how the joint system can maintain EVA's theoretical guarantees. Finally, overall evaluation will demonstrate that the joint system outperforms other methods with similar guarantees such as value iteration, and is also competitive with methods without such guarantees such as least squares policy iteration.

This chapter is organized as follows. First, I will cover training regimens in detail, explaining their use, and why they are well suited for EVA. I will then detail how to apply training regimens, explaining the two variations we developed and covering how careful application allows us to maintain EVA's theoretical properties. Finally, I will present empirical results, including user studies, looking at not only the effect of human input on EVA, but at how the joint system, EVA+, fares with respect to other MDP solution methods.

5.1 Training Regimens

To leverage human input with EVA, we introduce an interaction scheme we call *training regimens*. In this interaction scheme, users help the learning agent, *i.e.,* the machine, by providing a series of increasingly difficult problem instances. A problem instance refers to the problem of solving the MDP for a particular (starting) state, *aka.* the state specific MDP. We call this sequence of states a training regimen because it provides a course of study

for EVA. To our knowledge, the first published use of a training regimen comes from Asada in his 1994 paper on training a robot soccer player to shoot [9]. In this paper, Asada details how he set up a series of starting positions to speed up Q-learning. He began by placing the robot and ball close to the goal, and as the Q-learner improved, he would place the robot and ball farther, and farther away. Although Asada describes this process as a Q-learning optimization, it is essentially a training regimen in which the researchers themselves are the teachers. Since that time however, training regimens appear to have received relatively little attention. To our knowledge, it has never been studied as an interaction scheme for leveraging human input.

Training regimens can be seen as a method for guiding exploration. Unlike reward shaping where one guides exploration indirectly by authoring a potential-based shaping function [91], training regimens allow one to guide exploration directly by specifying states where the learner should focus. Training regimens pair well with EVA. The series of problem instances a training regimen provides serves to guide the expansions of EVA. Training regimens also do not solicit solutions, making them more efficient in their use of human time. Solutions provide little benefit to EVA due to its expansion based decomposition, which enables it to solve states to be expanded by leveraging the solutions of previous expansion rounds. This decomposition means it is typically faster for EVA to solve any given expansion state, than to ask for demonstrations.

Training regimens provide several benefits to the automated solver:

Focus: Automatic solvers typically assume the objective is to solve all states and that those states are equally important. These assumptions are often false. For example, we do not care about solving all chess board positions, just those reachable from the initial board. A properly-tailored regimen increases learning efficiency by maximizing generalization while minimizing the number of states the learner must see and solve. For example one may provide a higher density of examples in complex or important regions and fewer examples elsewhere. In an interactive setting, the states chosen can also be tailored to learner performance, *e.g.,* to highlight errors in the learned value function. Finally, proper focus provides better measures of performance as it allows us to weight errors based on the

relative importance of states.

Sampling: Approximation methods rely upon state sampling. Usually, the uniform distribution is used and it is assumed that the cost of sampling is low. Due to domain constraints, however, this assumption is not always true. Consider chess for example, while it may be easy to sample random board positions, it is much more difficult to sample just the reachable positions. Sampling also tends to be more difficult if we need to sample according to some specific distribution. EVA for example, requires the sampling of states based on their cost-to-gos. To deal with these difficulties, automated methods, including EVA, turn to rejection sampling. Unfortunately, depending on how well the envelope distribution matches the target distribution, rejection sampling can be very expensive to perform. A training regimen can remove the cost of sampling by providing needed states directly.

Decomposition: How states are ordered in a regimen guides the learner. By ordering states such that more difficult ones build upon simpler ones, we can save the learner significant work — solving a new state will then only require incremental effort. For pairing with EVA, the training regimen should be ordered to match, by cost-to-go. Specifically, states with low cost-to-gos should be placed first so that when EVA expands to higher cost-to-gos states, it can take advantage of prior solutions.

5.2 EVA+

Making use of training regimens in EVA is conceptually straightforward. In each round of EVA, instead of sampling some number of states to construct the training set for the function approximator, the states can be directly requested from the user. Importantly, this implies an interactive specification of the training regimen in which the user provides problem instances on a round by round basis. This interactivity enables the user to evaluate EVA's progress as it expands and adjust accordingly. For reference, the pseudocode for EVA is provided in Figure 46.

Unfortunately, while conceptually simple, it is impractical to require the user to specify all the states needed by EVA. Depending on the domain, EVA may require training sets of thousands of states or higher. To ask the user to specify so many states would be

Require: $M = (S, A, T, R, \gamma)$, D, L, r, ϵ_p, ϵ_l, δ
  // M - MDP to solve
  // D - sampling planner
  // L - learner (or estimator)
  // $r \in \mathbb{R} > 0$ - radius controlling the expansion size
  // $\epsilon_p \in \mathbb{R} \geq 0$ - planner precision parameter
  // $\epsilon_l \in \mathbb{R} > 0, \delta \in (0, 1]$ - precision parameters: must be accurate within ϵ_l,
  // with probability at least $1 - \delta$

  $\hat{J}_0 = \{$ **function** $s \rightarrow$ **if**(`absorbp`(s)) **then** 0 **else** $\perp$ $\}$
  $merp_0 = 0$
  for ($i = 1$; $\hat{J}_i$ is incomplete; i++) **do**
    $merp_i = merp_{i-1} + r$
    $C_s =$ # of samples needed by `L` to meet ϵ_l, δ precision
    $samples =$ sample C_s states within $merp_i$
    $examples = \{ (s, estJ(s)) : s \in samples, estJ(s) =$ `D`$(M, \hat{J}_{i-1}, merp_{i-1}, s) \}$
    $\tilde{J}_i =$ `L`$($ *trainset* $)$
    $\hat{J}_i = \{$**function** $s \rightarrow$
        $J_l, J_h =$ obtain $1 - \delta$ confidence interval on $\tilde{J}_i(s)$
        if $(J_h - J_l \leq 2\epsilon_l)$ **then** $\tilde{J}_i(s)$ **else** $\perp$ $\}$
  end for
  return $\hat{J}_i$

Figure 46: Pseudocode for the EVA algorithm. EVA generates a series of value functions, each more complete than the last and returns a final, fully defined value function. For purposes of disambiguation, we refer to L as the function approximator rather than as the learner. The pseudocode presented here is a copy of the original figure first presented in Chapter 3.

inordinately burdensome.

We address this difficulty in two parts. First, we avoid direct use of the training regimen to supply the necessary states. Instead we look to the regimen as a source of scaffolding, or supportive knowledge, on *where to sample*. The idea is to use the regimen to construct a better sampling distribution so that EVA can efficiently sample the states it needs itself. Second, we make the process of specifying a regimen for any given round interactive, and provide constant transparency into the accuracy of the approximator. In doing so, we can leverage the user to determine when enough data has been generated for a sufficiently accurate approximation, *i.e., when to expand*.

In the following discussion, I will cover this two part approach in detail. I will also present how careful application of these methods can enable EVA to maintain some, and sometimes all of EVA's theoretical guarantees.

5.2.1 Sampling Distribution Use

Rather than using training regimens to directly provide the states needed by EVA to expand, we can use them as a source of knowledge on *where to sample*. Specifically, we can use them to construct a distribution on the target region so that EVA can efficiently sample the states it needs itself. This can be accomplished by using the distribution directly as EVA's sampling distribution or as the envelope distribution for EVA's rejection sampling procedure. In either case, we call this approach to using training regimens, the "sampling distribution use" of training regimens.

Sampling distribution use can be viewed as a system for amplifying the effect of human help. When a user provides training regimen states, the system uses them to build a distribution over the state space on where to sample. As more states are provided, the system can begin to take over the generation of states. To the user, this process can be viewed as the system providing assistance by augmenting their input with automatic generation of regimen states. For this reason, we refer to sampling distribution methods as augmentation methods.

We explore two forms of augmentation. The first form is called the "exemplar form",

and the second is called the "distribution form". We follow with a discussion of the two forms with consideration of how they can be used in a way that maintains EVA's theoretical guarantees.

5.2.1.1 Exemplar form

The idea of the exemplar form of augmentation is for users to provide a few exemplar states and have the computer supply additional nearby states. Given an exemplar state, we have two mechanisms for generating nearby states. In the random walk mechanism, short random walks are performed from the exemplar state to generate additional states. The second mechanism, random perturbations, generates additional states by directly perturbing the feature vector that encodes the exemplar state. Some generated states may not be valid states or may be outside the target region of the expansion round, these states are discarded.

The exemplar augmentation form uses states in the training regimen as exemplars that implicit specify where to sample. In specifying the regimen, users can provide more exemplars in complex or more important regions of the state space and fewer exemplars in simple or less important regions. Specified properly, the set of exemplars can provide a high fidelity, non-parametric representation of the sampling distribution.

As a final detail, leveraging our results on interactivity, we structure the solicitation of exemplars so that it is interleaved with feedback from the system. We ask for exemplars one at a time, and after each given exemplar, show a small set of sampled states generated from that exemplar.

While empirical results in Section 5.3 show the exemplar form of augmentation to be effective at speeding up EVA, it also poses certain difficulties. In particular, we see that (1) providing exemplar states can be highly time consuming, especially for complex domains, and that (2) users tend to generate poorly distributed exemplars, and thus poor sampling distributions. The ladder difficulty can be particularly troublesome because poorly distributed training sets can lead to lower accuracies, prompting the need for larger training sets and retarding EVA's progress.

5.2.1.2 Distribution form

The difficulties of the exemplar form lead us to explore a second, alternative augmentation form, the "distribution form". In this form, instead of using the training regimen as exemplar states that implicitly specify the sampling distribution, the system requests that the user specify the training regimen through a set of constraints. Forcing the specification of the regimen in this way enables us to use the result as a more efficient envelope distribution for EVA's rejection sampling, giving EVA the ability to sample from the target region quickly and efficiently.

Specifying a training regimen in the distribution form is best explained through an example. Initially, users are given a training regimen generated from uniform sampling of the state space. The regimen will almost certainly be overly broad and contain numerous states outside the target region. The user is then asked to modify this training regimen through the specification of constraints to remove as many states outside the target region as possible without removing any that are within the region. In other words, the user is asked to sculpt the regimen through the use of constraints that restrict the types of states allowed. An example of a simple constraint for the grid-world domain might be a requirement for the `X`-coordinate of the robot to fall within some user-set range.

Forcing the user to specify the training regimen via constraints presents an important advantage: it ensures that the resulting regimen is some form of uniform distribution for which it is easy to construct a sampler. This enables EVA to use the training regimen directly as the envelope distribution for its rejection sampling procedure.

Interactivity for the distribution form is provided in a fashion similar to that used in the exemplar form. First, constraints are solicited one at a time. Second, both before and after a constraint is given, transparency into the efficiency of the training regimen is given. This transparency is provided by randomly sampling several states from the training regimen and showing whether they would be rejected or accepted.

The distribution form of augmentation, like the exemplar form, extracts knowledge of where to sample. However, it addresses the two key difficulties of the exemplar form.

First, it is typically simpler and faster for the user to give a few constraints on the envelope distribution than it is for them to give a set of exemplars to implicitly specify the sampling distribution. Second, the distribution form can only construct uniform sampling distributions. This limits flexibility, but prevents accidental specification of poor sampling distributions.

5.2.1.3 Maintaining Theoretical Properties

Using training regimens to construct EVA's sampling distribution must be performed carefully to maintain EVA's theoretical properties. Suppose, for example, the training regimen is used directly as the sampling or envelope distribution. In that case, the user becomes the sole source of samples. On the positive side, it enables users to focus EVA on just those states they deem important. On the negative side however, it can jeopardize EVA's guarantees. EVA expands in cost-to-go space and can only expand when *all* relevant states within the cost-to-go bound are well approximated. If the user misses some critical region of relevant states and prevents EVA from obtaining samples in that region, EVA may never be able to obtain a good approximation there. Such an event would either prevent EVA from expanding or let it expand with an inaccurate approximation. In either case, it would invalidate EVA's guarantees of convergence and optimality.

If we are careful in how the samples from the training regimen are used however, we can avoid this loss of guarantees. Specifically, we can do so by merging samples generated at the direction of the user with those generated from independent uniform sampling. For example, when a state is to be sampled, a coin can be flipped to see if the sample will be taken uniformly or based on the training regimen. This simple scheme ensures that all states have nonzero measure of being sampled, guaranteeing that with enough data, any appropriate function approximator will be able to construct a good approximation. Because the only impact of the sampling distribution is on the ability of the function approximator to obtain good approximations, and our simple scheme maintains this ability, the scheme will allow us to maintain EVA's theoretical results.

Using sample merging introduces certain inefficiencies. However, we note that it will

still allow us to derive significant benefit from training regimens. Consider the case when there is a 50-50 split between samples based on the training regimen and samples generated through uniform sampling. Let us suppose that the samples generated based on the training regimen have sample efficiency $1/k_1$ (*i.e.,* on average, one out of k_1 samples is usable), while samples generated uniformly have efficiency $1/k_2$. Then, if the function approximator needs n samples, the combined method will need at most $2\min(k_1 n, k_2 n)$ samples in expectation. In other words, the combined method will capture the majority of the benefits of the training regimen, assuming it is accurate, needing at most twice as many samples as using the training regimen alone. If the training regimen is inaccurate, the combined method will rely upon the uniform sampler, needing at most twice as many samples as without the training regimen.

Sample merging does have one significant downside: it precludes users from focusing on a subset of the state space. No matter what the user specifies, the uniform sampler it is merged with will ensure the inclusion of all states with the cost-to-go region of the expansion round. In cases when the user is certain of the accuracy of their training regimens, and significant speedups can be obtained by focusing on a subset of the state space, it may be fruitful to disable sample merging.

5.2.2 Expansion Timing Use

In using training regimens for determining where to sample, we enable EVA to sample the states it needs for itself, freeing the user from having to specify each needed state by hand. However, in doing so, we also lose the use of the regimen in determining when to expand. Fortunately, we can recover this expansion timing use by ensuring regimen specification is interactive and by providing continuous transparency into the accuracy of the function approximator. We provide transparency by periodically testing the approximator on test sets randomly drawn from the target region. In each testing, we display not just the accuracy of the learned hypothesis but the individual states the hypothesis was unable to accurate estimate. This provides the additional benefit of letting users see where and what kinds of errors the function approximator is making, giving them an opportunity to adapt their

sampling distribution by focusing on regions of high error.

Using human input for expansion timing is a mixed bag. On the positive side, it enables users to trigger expansions based on when they deem the function approximator to be sufficiently trained, which often leads to faster expansions and overall speedup. On the other hand, the danger is that users may tell EVA to expand too fast and jeopardize its guarantees.

To mitigate this danger, we can add error handling to EVA when user based expansion timing is used. Error handling for EVA is conceptually the same as error handling for ARVI, our expansion algorithm for acyclic domains. Whenever a state is sampled we can compare its predicted value with an independently solved value. For EVA this means using the planner. If the predicted value is x, EVA will call the planner with the approximator and heuristic value from round i where i is the highest round such that $merp_i < x$[1]. If the difference between the solved value and the predicted value exceeds twice the planner's precision, the state is considered an error. When this occurs, the solved value is used as the correction value and replaces the original. We also set the round back to i and set aside previously computed states with cost-to-go greater than $merp_i$. This essentially rolls back the algorithm to the beginning of the expansion round containing the error. We call this the "merp-regression" process. It forces cost-to-gos computed based on the erroneous value to reset, preventing propagation of the error. States set aside are added back in with each expansion as EVA resumes its normal behavior.

With error handling, we can regain some of the theoretical guarantees of EVA. In particular, we can guarantee that in finite domains, with an exact planner, the number of merp-regressions will be finite [148]. Thus, EVA will converge. Further, in the limit of infinite samples, if the approximator can represent J^* we can guarantee that EVA will converge to it. Additional details can be found in [148].

[1]Recall that $merp_i$ refers to the maximum effective radius of the predictor in round i, see Figure 46 for details.

5.3 Experiments

We explored EVA+ in two different domains. The first is a domain called Wall-E in which the objective is to move a cube of trash for disposal (see Section A.6 for details). Wall-E is the simpler of the two domains where we focused on evaluating the impact of human input. The second domain is a clone of the popular video game Super Mario Bros (see Section A.7 for details). Mario is a large and complex domain, used for annual AI competitions since 2009. In Mario, we focused on evaluating the overall EVA+ system as a whole. We compared the overall system to several well known baseline algorithms include value iteration (VI), least squares policy iteration (LSPI), and fitted-Q iteration (FQI).

5.3.1 Wall-E

The Wall-E domain is a simple game in which the agent, Wall-E, is tasked with moving a cube of trash from an initial location to a disposal location (see Figure 58). Wall-E must then return to a charger. Conceptually, Wall-E can be likened to a complex, scaled up version of the Taxi domain (see Appendix A.3). The Wall-E domain is discretized into a 12x12 grid. State is represented as the vector [`WallE-X`, `WallE-Y`, `CubeX`, `CubeY`, `Holdingp`, `PortX`, `PortY`, `ChargerX`, `ChargerY`]. `Holdingp` is a binary feature indicating whether Wall-E is currently holding the trash cube. Actions available to the agent are `North`, `South`, `East`, `West`, `Load`, and `Unload`. Rewards are uniformly -1 excepting the terminal state which has a reward of 0.

The primary focus in the Wall-E domain is in evaluating whether the EVA+ system can make effective use of human input. To do so, we looked at how well human trained policies performed as compared to those produced automatically by EVA. For leveraging human input, we focused on the exemplar augmentation form. We also used human input for determining when to expand. For EVA and EVA+, we used radius $r = 7.0$, an exact form of IDSS with precision $\epsilon_p = 0.01$, and the regression tree algorithm GUIDE [79] as the function approximator.

The results are compelling. Leveraging human input provided the experimenters, EVA+ is able to learn an optimal policy after just a few hours' time. By contrast, even after several

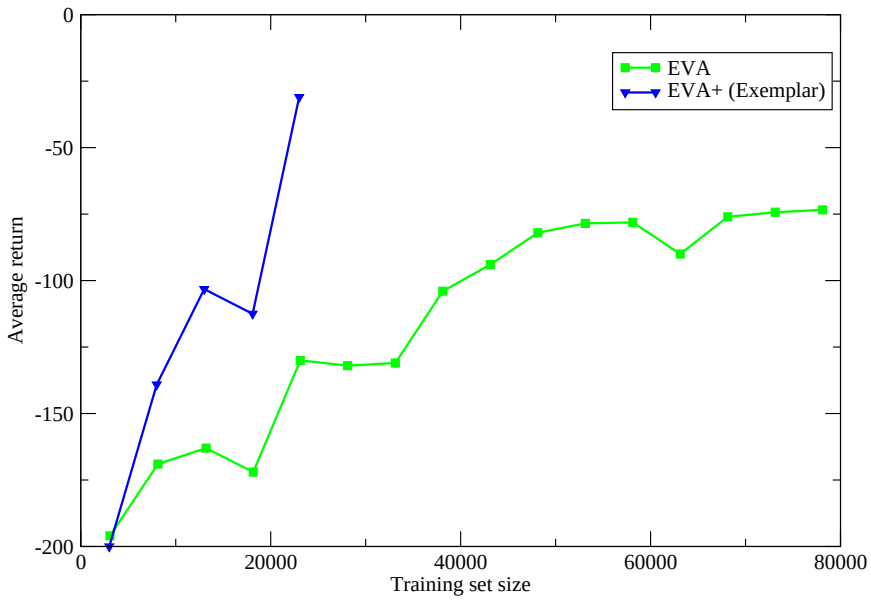

Figure 47: Learning sample complexity of EVA+ and EVA

days, EVA is unable to learn an optimal policy. Two factors drive the superior performance of EVA+, learning sample efficiency and rejection sampling efficiency.

EVA+ offers a higher learning sample efficiency than EVA. That is, given similar sized training sets, the one from EVA+ offers significantly higher performance than the one from EVA. Figure 50 shows the relative performance of EVA+ and EVA by training set size. Note how EVA+ is able to achieve optimal performance after just 24k examples, while EVA is unable to do so even after 80k examples. The higher sample efficiency of EVA+ suggests an ability to leverage human input to focus training on the important regions of the state space.

EVA+ also offers higher rejection sampling efficiency than EVA. In every expansion round of EVA, it must perform rejection sampling to obtain states within the target cost-to-go region. Because EVA uses a uniform envelope distribution, this rejection sampling can be costly when the target region is small. By contrast, EVA+ is able to leverage human input to focus its sampling. Figure 48 compares the efficiency of EVA+ and EVA over different expansion rounds. As we can see, EVA suffers the worst efficiency early on when the region of interest is relatively small, consisting of just the goal and its surrounding states. As EVA expands its approximation, the region of interest expands as well, yielding

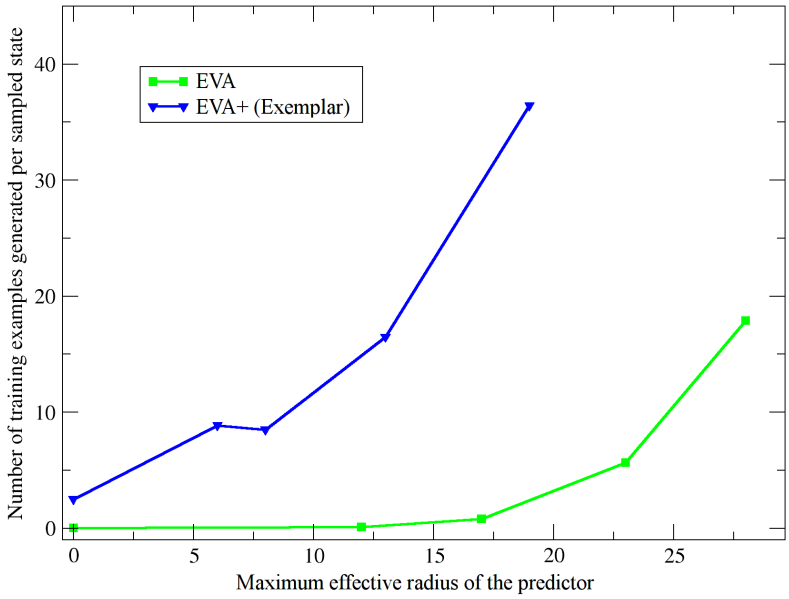

Figure 48: Rejection sampling efficiency of EVA+ and EVA

improved efficiency. This suggests human guidance to be more helpful at the beginning of learning.

5.3.1.1 User Study

Results thus far have been based on human input provided by researchers. To study whether end-users can enjoy similar success, we performed a user study soliciting participants from the campus community. In our user study, we cast the EVA+ system as a video game using a school metaphor. The participant takes on the role of a teacher whose job is to advance the learner (or student) through successive grades until "graduation", when the learner can successfully perform the target task. Participants are provided an interface where they can give the learner a series of states organized as "homeworks". Homeworks represent the states for EVA to solve and label to generate the training set it needs to expand the approximation region. Teachers are also prompted to give "tests", sets of states that are previously unseen by the learner to measure its progress. Tests are a transparency mechanism for providing insight into whether the function approximator has sufficient training data to accurately expand the approximated region. When the teacher deems the learner's performance sufficient, they can advance the agent to the next grade. Finally, the agent

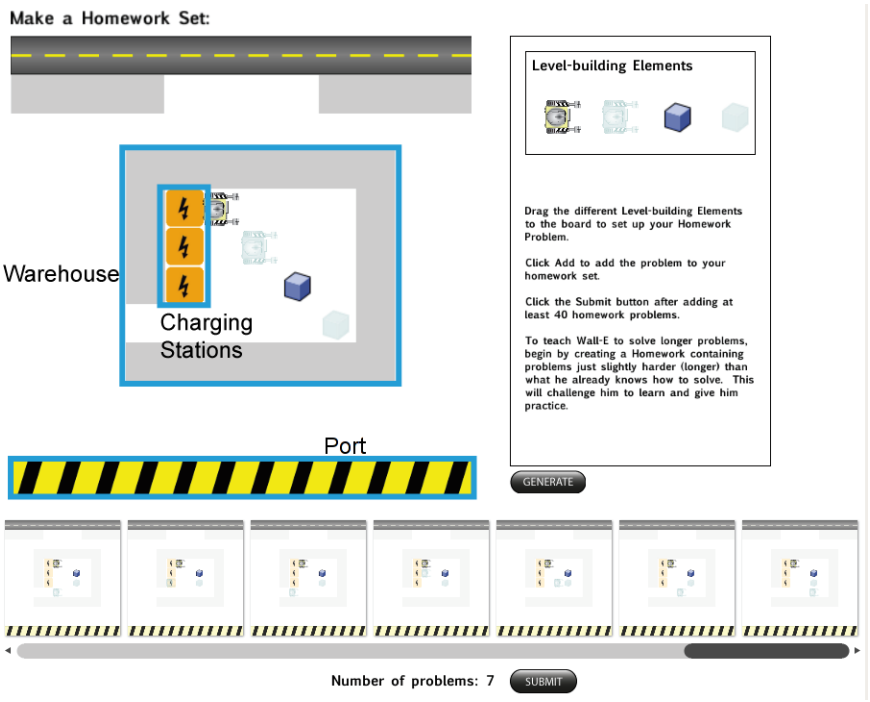

Figure 49: The homework creation screen of our game. The right panel holds our level building elements and instructions. The bottom panel collects the problems generated. The main element of the screen is a level map showing initial and end locations (ghosted icons)

may "flunk" out of a grade if they are advanced prematurely. This occurs when the EVA+ system finds an error in the learned value function and performs a merp-regression to correct the error and to prevent it from propagating.

Figure 49 shows our game in the homework creation screen. In this screen, the teacher is asked to create states which, together, will make up a homework. Once a homework has sufficient states and is considered complete, the teacher can press the "submit" button to give the homework to Wall-E to practice on. We use the game metaphor of a "level editor" for creating states where each level corresponds to a state. To create a state (or level), the human teacher simply drags and drops various level building elements such as Wall-E's start and end positions and the cube's start and end positions on to the board. Once the level is setup, the teacher must click the "Generate" button. This adds the state and many similar ones to the homework. Generation of similar states represents the exemplar augmentation at work, and enables the teacher to create more than one state per board setup.

Recall that states must be ordered by increasing cost-to-go. Wall-E's current "grade"

controls the cost-to-go range that is accepted. In other words, the grade corresponds to the round of EVA. We enforce states of increasing cost-to-gos by starting the game with Wall-E in the lowest (1st) grade. In this grade, only states with low cost-to-gos are accepted so the human teacher must provide problems that are almost already fully solved. To advance Wall-E and gain access to more difficult problems, the teacher must give him homework(s) and evaluate his progress via tests. In a test, states that are accurately estimated are considered to be "correct" while states that are not are considered to be "wrong". A high test score (*i.e.,* a high percentage of correctly estimated states) implies that sufficient training data has been generated and that EVA is ready to move to the next expansion round. Only at this point can the teacher advance Wall-E to the next grade.

In our study, participants were first given instructions and left alone a few minutes to read them. The instructions begin by providing some background, introducing the user to their role as a teacher and the school-based context of the game. It then explains that their objective is to teach Wall-E and advance him through the grades until he can complete the target task. To help encourage motivation in the participants, we hold a competition among the trained Wall-Es. The trainer of the winning Wall-E, as the instructions explain, wins a gift card. The rest of the instructions describe the operation of various game interfaces and briefly explain the expected progression. Namely, that Wall-E starts in 1st grade where he cannot solve but the "shortest" of problems, but will be able to solve "longer" problems as he advances grades. The instructions also warn that advancing too quickly may result in Wall-E learning incorrect concepts which may cause him to flunk out of later grades. To help ensure that participants understand the instructions, the experimenter also provided a brief demonstration highlighting the more important interface elements. Participants were then left to play the game. At the end of the study, an exit survey and brief interview were conducted. Due to time constraints, we did not require participants to complete the game. Instead, we allotted a maximum of one hour for each participant. Participants could also choose to stop early.

We had 10 participants for our study. Participants were drawn from the campus community. Their ages ranged from 16 to 54. Education levels as reported on the exit survey

ranged from high school to PhD students.

To measure how effective participants were at using EVA+, we looked at how well their trained Wall-Es performed and the number of training examples needed to reach that level of performance. While participants gave varying amounts of training to Wall-E, on average, they gave 4,500 training examples whereupon Wall-E is able to solve approximately 15% of randomly selected problems. By contrast, under similarly sized training sets, the Wall-E trained by experimenters could only solve around 7% of randomly selected problems. Surprisingly, it appeared that not only were end-users capable of providing effective training regimens, but that they were more efficient at doing so than the experimenters. Analysis of the results reveal why. When we, as experimenters, gave training examples, we had a greater awareness of potential merp-regressions and tended to err conservatively to avoid them. As a result, we tended to give many examples before advancing. By contrast, users tended to advance earlier and simply retrained when faced with regressions. This more aggressive approach led to higher sample efficiency. These results lead us to conclude that end-users are capable of providing effective training regimens.

5.3.1.2 Speeding up the Interaction

While overall results are suggestive of the potential of training regimens, the amount of interaction time required is a major concern. In the experiments, it took several hours to train Wall-E to completion.

Analysis shows two major sources of time consumption. First, construction of exemplar states is slow. Each state construction is a level creation task which can take a significant amount of time even for a simple game such as Wall-E. When combined with the fact than many exemplar states are needed to train EVA+, it becomes a major drain on users' time. Second, the training data distribution tends to be uneven. Uneven training data retards expansions in EVA because it can lower function approximation accuracy, prompting the need for larger datasets. This not only means more computation required to generate a larger dataset, but more training time required by the function approximator. Further, larger datasets can require more exemplar states which can exacerbate the previous problem.

The unevenness of the training data distribution is particularly interesting. In the study, several participants reported difficulty coming up with enough states to give the system although they could roughly specify the region of interest. This leads us to conclude that users may have difficulties sampling from multi-dimensional regions such as those encountered in Wall-E. This conclusion is supported by two facts. First, study participants were not given access to pen and paper or other secondary storage, so it would be difficult for them to sample systematically. Second, from research in psychology we know that humans tend to have difficulty generating random samples [138, 39].

To avoid these difficulties and shorten the amount of interaction time required, we turned to the distribution form of augmentation. The results are encouraging. Unlike when we used the exemplar form of augmentation, much of the time in training EVA+ under the distribution form of augmentation was spent waiting for the learner to provide feedback. Once a good set of constraints were given to EVA+, we could simply wait until the system generated enough examples to accurately approximate the target region. This significantly shortened the amount of users' time required. Additionally, the distribution form of augmentation was more sample efficient, cutting the overall training time by over one half. Figure 50 compares the sample complexity of the two forms of augmentation. As we can see, removing the unevenness of the exemplar form of augmentation can yield significant benefits and appears well worth the loss of flexibility.

5.3.2 Mario

The Mario domain (see Figure 59) is a complex game based on the classic video game, Super Mario Bros. In Mario, we focused on evaluating the overall EVA+ system as a whole.

Our Mario implementation is based on that of the Mario world used in the Mario AI competition. On face value, the game is partially observable and has a series of fixed levels. Further, in Mario AI, levels are procedurally generated and can be infinite. To use Mario as a domain for our purposes, we must model it as an MDP. As we are interested in a model which will let us solve the game in general as opposed to one which will solve a specific level of fixed length, we use the Mario screen as the game. In other words, for purposes of

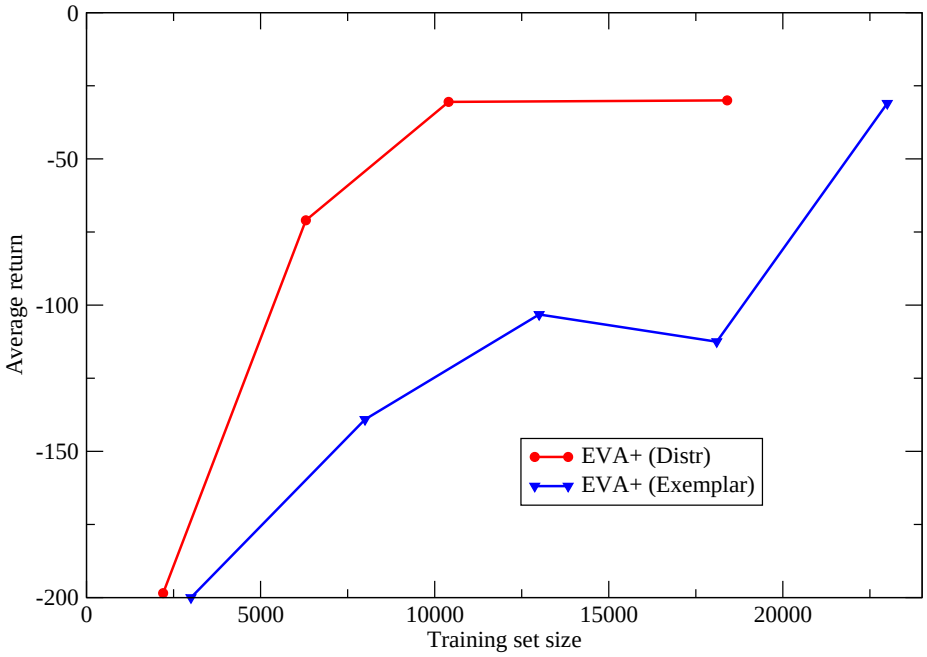

Figure 50: Sample complexity of different variants of EVA+

modelling, we treat a screen of Mario as the entire game in which reaching the right edge of the screen is synonymous with winning. Modelling in this way allows us to construct an agent that can solve any Mario screen which should be sufficient to perform well in the actual game.

In our Mario MDP, the state space consists of the list of all visible objects and their properties. In other words, it includes all coins and their locations, all enemies and their locations, velocities, and states, all mushrooms and their locations and velocities, *etc.* The state space also includes the map of all screen tiles, *i.e.,* hills, blocks, gaps, *etc.* There are fourteen actions available: `Noop`, `Shoot`, `Jump`, `Jumpshoot`, `Left`, `Leftshoot`, `Leftjump`, `Leftjumpshoot`, `Right`, `Rightshoot`, `Rightjump`, `Rightjumpshoot`, `Down`, and `Downshoot`. In Mario AI, no reward function is specified. Instead, agents are scored based on average distance traveled. Ties are broken first by number of enemies killed and number of coins gained, and then by non-functional considerations such as running time. We chose a reward function which roughly mirrors this goal: killing an enemy (10), coin (20), mushroom (60), fire flower (60), getting hurt (-40), winning (7000).

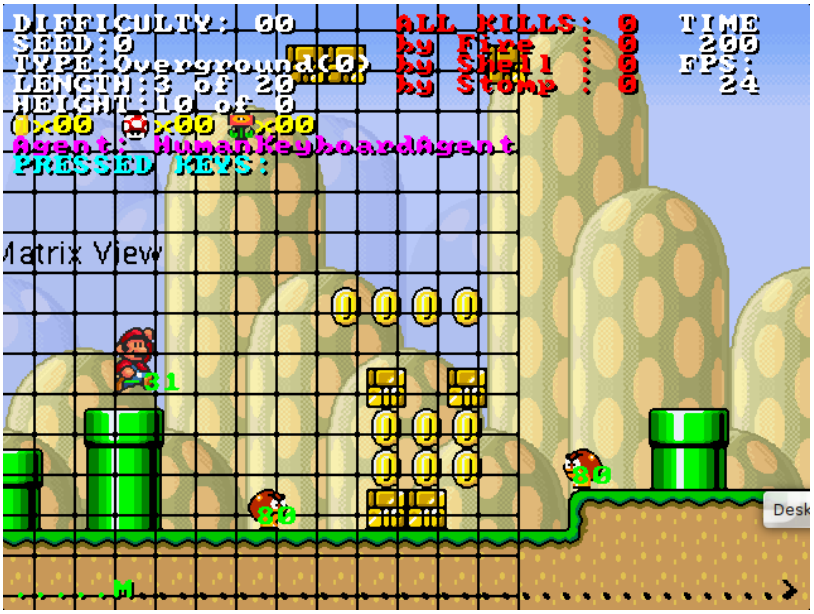

Figure 51: The Mario domain

Learning from the difficulties of the exemplar augmentation form, we focus on the distribution form for the Mario domain. Here we also use the training regimen more conservatively. Sampling is not performed directly based on the training regimen but rather in conjunction with uniform sampling. The training regimen is also not used to decide when to expand. These choices reflect, in part, our desire to study the benefits of training regimens under more conservative use, when it maintains all of the theoretical properties of EVA. However, they also reflect practical limitations stemming from the complexity of the domain. In Mario, the domain is complex and the transition function is costly to compute. Planning in Mario is slow. Searching forward even just a few steps quickly leads to an explosion in the size of the planning tree. As a result, EVA+ cannot provide feedback to user inputs in real time. A single expansion round may take over a day. Thus it is impractical to ask users to monitor the accuracy of the function approximator to decide when sufficient examples have been generated.

In Mario we used radius $r = 21$, an exact form of IDSS with precision $\epsilon_p = 0.1$, and linear regression as the function approximator with maximum standard deviation $\sigma_{max} = 1$.

In the Mario user study, we followed the same school metaphor as that used in Wall-E. EVA+ takes the role of the learner or student, and the user takes the role of the teacher.

However, the similarities stop there. Unlike in Wall-E where the student is given "problems" (*aka.* start states) to practice on, in Mario, the student is actively trying to find these problems. Problems range in difficulty falling into one of four categories: easy, medium, hard, and impossible. The student is trying to find a good mix of easy, medium, and hard problems to work on. The difficulty for the student is that they do not know where to look. All they can do is pick random states and hope to get lucky. The participant takes the role of a teacher whose job is to tell the student where to look so they can find good practice problems more quickly. However, they must take care not to exclude good problems with their instructions or the student will not learn well.

Ideally, the process of leveraging human help would be done in the larger context of a student that improves over time and moves from simpler to more difficult problems. In Mario however, because expansions are highly time consuming, each participant focuses on just one expansion round and is bereft of the larger context. Importantly, participants have no way of knowing what Mario can and cannot already solve. To resolve this difficulty, in our study, the participant is first asked to observe the student as he sorts through randomly chosen problems into piles of easy, medium, hard, and impossible problems. Only then is the participant asked to provide guidance on where to search. This added observation step offers a limited form of interactivity in which transparency is provided prior to user guidance, but not afterwards. This limits the dual-channel effect of interactivity — discussed in Section 4.3 — to just a single channel. As a result, users can tailor their guidance based on the observed state of the learner, but cannot obtain feedback on the effects of their guidance to adapt accordingly.

We asked participants to provide guidance in the form of constraints on what states can be generated. Participants provide constraints by filling out a simple form which lists every object type in Mario and the properties of that object type. For example, the object type might be "coins" and its properties might include items such as the `X` location of the coin, and the `Y` location of the coin. Each item is given a large box labeled with its range of possible values. For example, the `X` location box for coins might be labelled by the range [0, 320]. A constraint is specified by filling in a box with a set of possible values which

instructs the student where to look. Thus a participant might fill in the `X` location box for Mario with 240-260, which would instruct the student to focus their search to just those problems in which Mario starts with its `X` position in that range. An unedited box indicates a lack of any special instructions to the student.

We performed the user study with nine participants. We began by explaining the instructions and demonstrating several examples. The participants were then allowed to observe the student as they sorted through randomly chosen problems into piles of easy, medium, hard, and impossible problems. Whenever they indicated readiness, we provided them with the form to provide constraints to the student. Participants were told to narrow the search of the student as much as possible, but that it was critical that they not remove any potentially solvable problems. Participants were also told that they need not fill in every box or even any boxes.

To measure the effect of our users' input, we compared the performance of EVA+ to that of its automated counterpart, EVA. While both algorithms converged to a near-optimal policy, EVA+ was able to do so with half as many sampled states and a 13x reduction in computation time. Analysis shows that EVA+ does not train its approximator with fewer examples. Rather, the gains come from increased efficiency in rejection sampling. When measured in computation time, the improvement is even greater. This is because when EVA+ increases the efficiency of rejection sampling, it does so by removing samples that take the longest to evaluate which are those determined as too hard to solve. We did not perform comparisons with the exemplar augmentation form. Mario states are so complex that generating even a few states can become time consuming.

Next, we study the results of EVA+ and EVA in a larger context. We compare EVA/EVA+ to theoretically convergent and optimal methods such as VI, as well as modern function approximation methods such as LSPI and FQI although they lack theoretical guarantees of convergence and optimality. Note that VI could not complete due to the size of the state space in Mario and had to be truncated.

We first focus our comparison on solution quality. To ensure an even comparison,

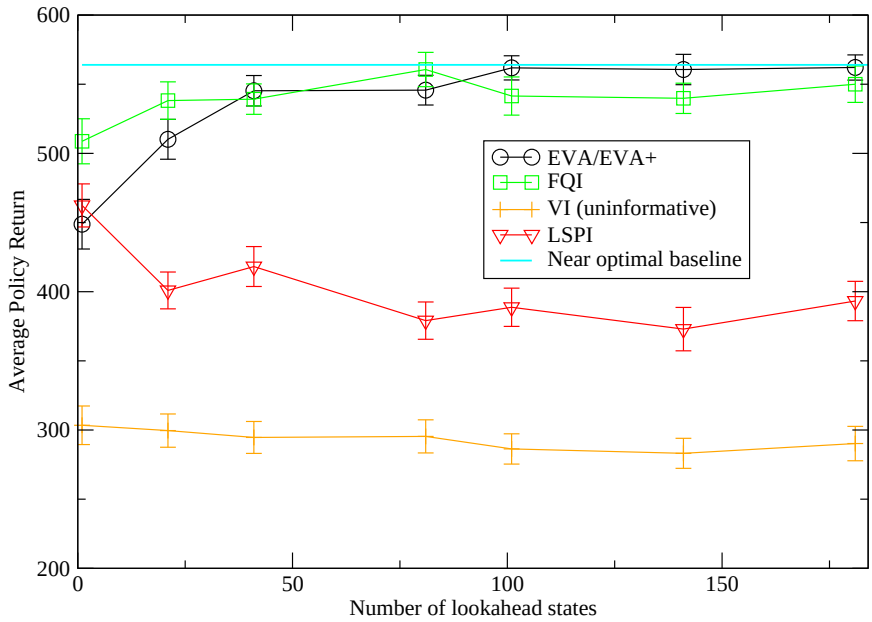

Figure 52: Solution qualities of EVA/EVA+, VI (truncated), LSPI, and FQI, over varying amounts of lookahead. Policies are cut-off after a maximum of 100 steps. Results show averages of 100 evaluations. Error bars denote one standard deviation from the mean.

all approximate methods are given the same linear model with the same state features[2]. Figure 52 shows the average return of the k-greedy policy for the various methods over different ks. A k-greedy policy for a state s is formed by performing lookahead search starting from s until k states have been expanded. At that point, the estimated best action is returned. Ties are broken arbitrarily.

From the figure, two points of interest are immediate. First, the solution quality of EVA/EVA+ is competitive with that of other methods. FQI performs slightly better under low lookahead conditions, however, the difference quickly disappears with increased lookahead. FQI is an action-value method with additional features for representing various actions and this may be partly responsible for its initial performance. LSPI, being an action-value method as well, also performs competitively under low lookahead. However, unlike FQI, its performance deteriorates with increased lookahead. This leads to a second point of interest: deeper lookahead does not appear to help LSPI as it does EVA and FQI. A closer examination reveals why. EVA estimates, even when they are inaccurate for a

[2]LSPI and FQI are given additional features, one dummy variable for each action, because they are action value algorithms rather than state value algorithms

state, tend to be close to the true, absolute cost-to-go. Since forward search enables one to skip past small mistakes and find the right general direction, increasing the search depth enables better policies for EVA. FQI estimates tend not to be as close to the true, absolute cost-to-gos; they tend to be off by a translation. However, FQI estimates tend to maintain, at least regionally, the shape and gradient of the value function. As a result, forward search can still be helpful in avoiding local mistakes and yield improved performance. LSPI estimates are a different matter. LSPI estimates of different action values within a state tend to be accurate relative to each other. However, estimates of values over different states are not always close to the absolute cost-to-go, nor do they always maintain the regional shape and gradient of the value function. As a result, forward search can sometimes have a detrimental effect. The effect is not always observed; the solutions produced by LSPI have great variability. The best runs of LSPI are competitive with FQI. The worst runs of LSPI can show degradation of performance to near random policy levels. The results presented in Figure 52 represent the performance of a median run of LSPI.

Next, we compare the algorithms in terms of computational cost. Although EVA produces competitive policies, its cost is significantly greater than that of LSPI and FQI. EVA took a little over ten days to compute the policy it returned. EVA+, taking advantage of human help, took less than a day, excluding the time it took to run the study. By contrast, however, both LSPI and FQI completed within ten to twenty minutes. The high cost of EVA/EVA+ reflects the high cost of its sampling planner. To generate the roughly 16k training examples used to build the approximate value function, EVA/EVA+ performed over 2 billion sample backups made over the course of 55k calls to the sample planner.

5.4 Summary

In this chapter, we combined our work on approximation methods (Chapter 3) with our work on leveraging human input (Chapter 4). Specifically, we developed a human interaction scheme we call "training regimens" for use with the approximation method, EVA. Training regimens can be likened to demonstrations, but without solutions; they contain just a series of starting states. Training regimens work by guiding and focusing exploration, and by

providing an implicit decomposition through their ordering. By ordering states by cost-to-go, training regimens become ideally suited for EVA and can be used to directly provide EVA with the sample states it needs to expand.

Because very large training regimens are usually necessary for learning, direct use of training regimens is often impractical. Instead, we apply our results from Chapter 4, and look to training regimens as a source of scaffolding on where to sample and when to expand. We explored two variations of this approach, the exemplar form and the distribution form. While we found both to be effective, the latter is often preferable because it puts a lower burden on the user. We also showed how these methods can be used with EVA without disrupting its theoretical properties by tempering its use with uniform sampling.

To demonstrate the effectiveness of combining human input with EVA, we tested the joint system, EVA+, on two domains. The first is a taxi-like domain called Wall-E, while the second is a clone of the popular video game, Super Mario Bros. In both domains, user studies showed EVA+ to be effective at leveraging help from end-users to obtain significant speedup over its purely automated counterpart, EVA.

Broadening the evaluation, we compared EVA+ to a variety of alternative MDP solution methods. The results are promising. When compared to methods which offer guarantees of convergence and optimality such as VI, EVA+ dominates. In fact, owing to the large size of the Mario state space, VI could not complete at all. When compared to methods which do not offer such guarantees including function approximation methods such as LSPI and FQI, EVA+ is competitive. Although EVA+ has a significantly higher computational cost, the quality of its solutions match or exceed that of other methods.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

This dissertation demonstrates that approximate methods, in conjunction with human input, can solve large MDPs while maintaining optimality and convergence guarantees. In this chapter, I will provide a summary of the methods and algorithms presented in the dissertation, and present some directions for future research.

6.1 Summary

In this dissertation I explored methods for solving MDPs scalably. I took a two-pronged approach:

A) I focused on how to use approximation in MDPs without losing convergence and optimality guarantees.

B) I focused on how to leverage human input, specifically how input can be used while being robust to errors, and how input can be solicited to maximize speedup and user engagement.

In Chapter 3, we tackled the problem of building an effective approximation algorithm with good convergence properties. We showed that the key difficulty that arises in approximation is the use of bootstrapping, and that by avoiding it we could obtain convergence and optimality guarantees. To avoid bootstrapping, we took an alternative tack we called the "expanding approach" which constructs a series of value functions that cover ever larger portions of the state space. We described two different expansion strategies, which led to two different algorithms ARVI and EVA. ARVI makes a compromise between expansion and bootstrapping. It is fast, easy to understand, very effective under tabular representations, but limited to acyclic domains. EVA uses sampling planners, which are, in the worst case, exponential in the horizon of the problem. However, unlike ARVI, it is generally applicable. Most importantly, EVA has good theoretical guarantees. EVA is guaranteed to converge to

a solution that can be probabilistically bounded *wrt.* the optimum. The bound is also well behaved — it is linear in the number of rounds.

In Chapter 4, we focused on methods of effectively leveraging human input. First, we studied how to use human input and showed that extracting supportive knowledge from human input rather than using it directly as examples to imitate is more effective and enables us to insulate against potential errors. Specifically, we explored two different methods. The first used demonstrations to discover subtask decompositions, while the second used demonstrations to find relevant features. In both cases, we saw that policies learned in the indirect fashion not only outperformed directly learned policies, but the users themselves. We then studied how to solicit human help and showed that interactivity plays an important role in terms of learning performance as well as teaching experience. The results also suggest that learning algorithms that can show consistent, significant improvements will be more effective at obtaining human help.

In Chapter 5, we combined these two prongs and developed a human interaction scheme we call "training regimens" for use with the approximation method EVA. The resulting system, EVA+, takes the best of both worlds. First, with careful use of training regimens, EVA+ inherits the theoretical guarantees of EVA. Second, user studies show that EVA+ allows end-users to obtain significant speedups. EVA+ also compares favorable to both existing tabular methods such as VI as well as existing approximate methods such as LSPI. In the Mario domain, EVA+ is able to learn the highest performing policy compared to those learned by LSPI and truncated VI [1]. We do note, however, that EVA was computationally more expensive than LSPI.

6.2 Future Work

My work has focused on approximation in MDPs and use of human input. I presented several methods and analyses of these methods to this end, but many improvements and much more exploration remains. In the following subsections, we will consider some of these improvements and explorations in turn. Finally, looking ahead, we consider some future

[1] VI could not complete due to the size of the state space

directions.

6.2.1 Approximation in MDPs

We presented two different expansion strategies for performing approximation in MDPs. Both are imperfect. In particular, EVA has worst case complexity that is exponential in the horizon of the problem. Although in practice we rarely see this worst case, EVA is nevertheless computationally expensive. One possible approach to improving the speed may be to use an adaptive radius, allowing EVA to expand more slowly over difficult regions but quickly over easier regions. Another possibility is to use standard bootstrapping solutions within EVA to solve certain state regions that are too expensive for sampling planners. Alternatively, one could use EVA in a bootstrapping framework and solve the same MDP over increasing horizons until the desired horizon is reached. On the whole, many possibilities and various compromises between the expansion and iterative approaches have yet to be explored.

6.2.2 Using Human Input

We have only scratched the surface of understanding how to apply human input. For example, leveraging human input under approximate methods is a very different problem than under exact ones. We have presented one method, training regimens, but others remain unexplored. For example, human input could be used to create problem relaxations, or to construct new features, or to aid in transfer learning. At a high level, the myriad of results from how humans teach each other and from curriculum design could be applied.

We have, thus far, limited our focus to explicit solicitation of human input. In these situations, the target task is well defined and the user is actively participating in the role of a teacher. However, the amount of time humans spend in such a structured teaching environment is far outweighed by the amount of time spent focused on other activities. Yet, little exploration has been done on non-intrusive acquisition of human help, that is, the leveraging of information from observing humans through the natural course of events, outside well-defined tasks and structured teaching interactions. If nothing else, this source of information can be used to better leverage active human help and to reduce the amount

of attention needed.

6.2.3 Future Directions

One of the benefits of the approximation work presented here is that it opens up the MDP formalism to different representations. This direction naturally leads to the question of what representations should be used and more importantly, how to learn appropriate representations. I believe the key will lie in the decision making context. Unlike in prediction settings, the learner in a decision setting has actions it can take to interact with the environment. Work in this direction must focus on building representations, including rich representations of actions. Our hope is that by showing how approximation can be used without losing convergence properties, we will enable a greater focus on learning rich, approximate representations.

APPENDIX A

DOMAINS

A.1 Mountain car (MCAR)

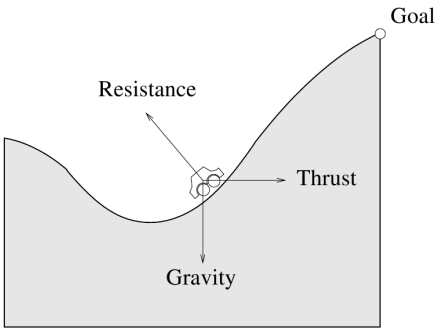

Figure 53: Mountain car domain. Graphic courtesy of [120]

Mountain car (MCAR), see Figure 53, is a two-dimensional, continuous, control problem. A car must rock back and forth in a trough until it gains enough momentum to carry itself atop a hill. All rewards are zero except for the final reward for succeeding at the task by reaching the top of the hill, which is one. The state space is a combination of position and velocity. There are three possible actions: full throttle forward, full throttle reverse, and zero throttle.

A.2 Pendulum domains

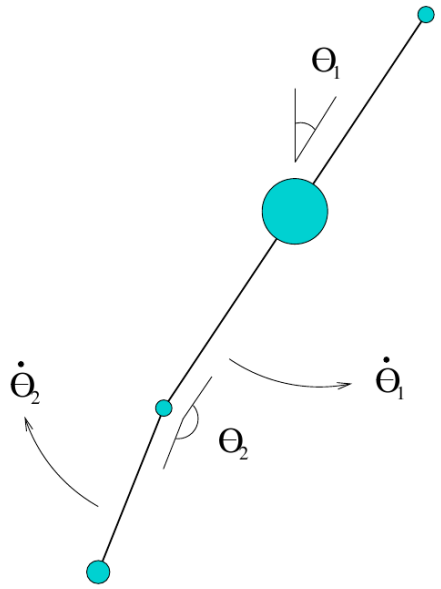

Figure 54: Double-arm pendulum. Graphic courtesy of [142]

The single-arm pendulum (SAP), (see Figure 54 for the similar double-arm pendulum domain), is a two dimensional, continuous, optimal control problem. The objective is to learn to swing up the pendulum and balance it. The agent has two actions available, a positive and a negative torque. Like in the mountain car problem, the torques available are underpowered, requiring swinging of the pendulum to gather momentum. Reward is set to zero for all states, except those in the balanced region, which give a reward of one. The state space is composed of the angle of the link and its angular velocity.

The double-arm pendulum (DAP), is a two-link (and thus four-dimensional) variant of SAP. Reward is set up similarly. The state space is composed of the angle and angular velocity of each of two hinges.

A.3 Taxi

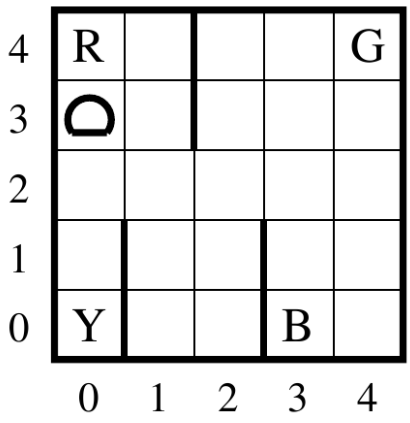

Figure 55: Taxi domain. Graphic courtesy of [34]

Taxi domain describes a world in which an agent is a taxi whose objective is to move a passenger from a starting location to a desired destination. The world is a 5×5 grid. There are 4 pickup/dropoff locations each residing in one of the four corners of the grid: `NW`, `NE`, `SW`, and `SE`. The state space is composed of three state features: `taxiLocation` `dropoffLocation` and `passengerLocation`. The actions are *North*, *South*, *East*, *West*, *Pickup* and *Dropoff*. The MDP terminates when `passengerLocation` equals `dropoffLocation`. Reward is uniformly -1 except for invalid *Pickup* and *Dropoff* actions which yield -10.

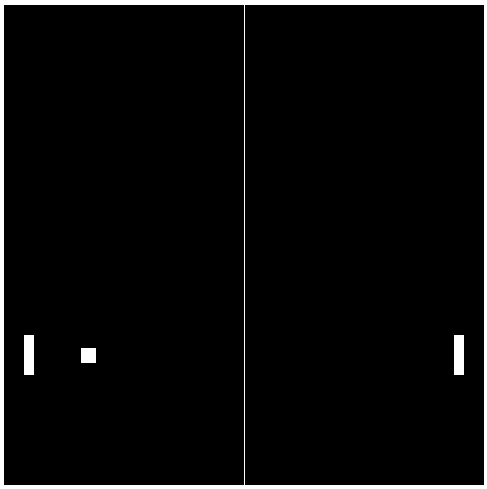

Figure 56: The Pong domain

A.4 Pong

Pong, see Figure A.4, is a game like table tennis in which two paddles move up and down to keep a ball in play. The paddle that misses the ball and allows it to off-screen loses. The agent uses one paddle while the other is controlled by a fixed policy which moves in the direction to best match the ball's Y position when the ball is approaching, moving randomly otherwise. There are five state features: `paddle-Y`, `ball-X`, `ball-Y`, `ball-angle`, and `opponent-Y`. `Y` coordinates and `ball-angle` have 24 possible values while `ball-X` has 18. There are two possible actions: `Up` or `Down`. Reward is 0 except when successfully returning a ball, yielding +10. The game terminates when a player loses or after 400 steps, implying a maximum policy return of 60.

Figure 57: Capture of the Frogger domain.

A.5 Frogger

This domain is a version of the classic Frogger game (see Figure 40). In the game, the player must lead the frog from the lower part of the screen to the top, without being run over by a car or falling into the water.

The screen is divided in a grid, and the state features are the contents of each cell relative to the current position of the frog. For example, the feature `3u2l` is the cell three rows up and two columns to the left of the current frog position, and the feature `X1r` the cell just to the right of the frog. The possible values are `empty`, if the cell falls out of the screen; `good`, if the cell is safe; and `water`, `carR`, `carL` for cells containing water, or a car moving to the right/left. Frogger has 5 possible actions: `Up`, `Down`, `Left`, `Right`, and `Wait`. Rewards are $r = 100$ for reaching the goal, $r = -100$ for death, and $r = -1$ for any other action.

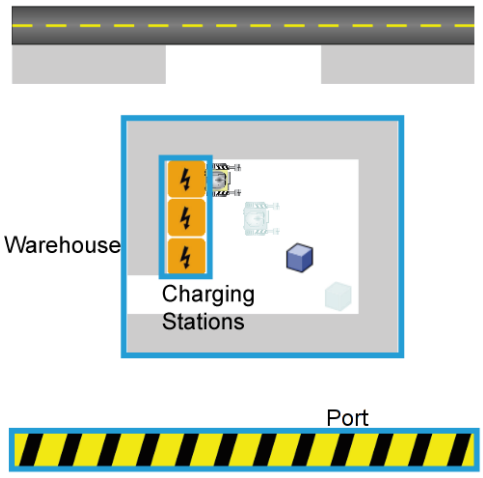

Figure 58: The Wall-E domain

A.6 WallE

The Wall-E domain is a simple game in which the agent, Wall-E, is tasked with moving a cube of trash from a warehouse to a port for disposal (see Figure 58). Wall-E must then return to a charging station. Conceptually, the Wall-E domain can be likened to a complex, scaled up version of the Taxi domain (see Appendix A.3).

The Wall-E domain is discretized into a 12x12 grid. State is represented as the vector [`WallE-X`, `WallE-Y`, `CubeX`, `CubeY`, `Holdingp`, `PortX`, `PortY`, `ChargerX`, `ChargerY`]. `Holdingp` is a binary feature indicating whether Wall-E is currently holding the trash cube. Actions available to the agent are `North`, `South`, `East`, `West`, `Load`, and `Unload`. Rewards are uniformly -1 excepting the terminal state which has a reward of 0.

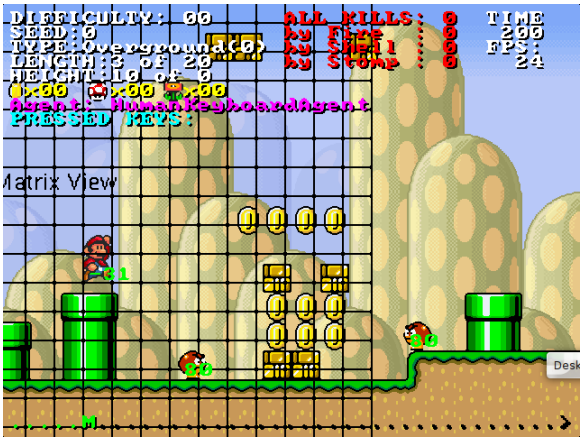

Figure 59: The Mario domain

A.7 Mario

The Mario domain is based on the classic video game, Super Mario Bros. It models a single screen of the video game. In other words, for purposes of modelling, we treat a screen of Mario as the entire game in which reaching the right edge of the screen is synonymous with winning.

In the domain, the state space consists of the list of all visible objects and their properties. In other words, it includes all coins and their locations, all enemies and their locations, velocities, and states, all mushrooms and their locations and velocities, *etc.* The state space also includes the map of all screen tiles, *i.e.,* hills, blocks, gaps, *etc.* There are fourteen actions available: `Noop`, `Shoot`, `Jump`, `Jumpshoot`, `Left`, `Leftshoot`, `Leftjump`, `Leftjumpshoot`, `Right`, `Rightshoot`, `Rightjump`, `Rightjumpshoot`, `Down`, and `Downshoot`. Rewards are +10 for killing an enemy, +20 for picking up a coin, +60 for eating a mushroom, +60 for picking up a fire-flower, −40 for getting hurt, and +7000 for winning.

REFERENCES

[1] Abbeel, P. and Ng, A. Y., "Apprenticeship learning via inverse reinforcement learning," in *International Conference on Machine learning*, (New York, NY, USA), p. 1, ACM, 2004.

[2] Abbeel, P. and Ng, A. Y., "Exploration and apprenticeship learning in reinforcement learning," in *International Conference on Machine Learning*, (New York, NY, USA), pp. 1–8, ACM, 2005.

[3] Aha, D., Molineaux, M., and Ponsen, M., "Learning to win: Case-based plan selection in a real-time strategy game," *Lecture notes in computer science*, vol. 3620, p. 5, 2005.

[4] Aler, R., Garcia, O., and Valls, J., "Correcting and improving imitation models of humans for robosoccer agents," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 3, pp. 2402–2409, IEEE, 2005.

[5] Argall, B., Browning, B., and Veloso, M., "Learning robot motion control with demonstration and advice-operators," in *International Conference on Intelligent Robots and Systems*, vol. 8, Citeseer, 2008.

[6] Argall, B., Chernova, S., Veloso, M., and Browning, B., "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[7] Argyle, M., Ingham, R., Alkema, F., and McCallin, M., "The different functions of gaze," *Semiotica*, vol. 7, no. 1, pp. 19–32, 1973.

[8] AS, A., "Nonverbal behavior and nonverbal communication: What do conversational hand gestures tell us?," *Advances in experimental social psychology*, p. 389, 1996.

[9] Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K., "Vision-based behavior acquisition for a shooting robot by using a reinforcement learning," Citeseer.

[10] Atiya, A., Parlos, A., and Ingber, L., "A reinforcement learning method based on adaptive simulated annealing," in *Micro-NanoMechatronics and Human Science, 2003 IEEE International Symposium on*, vol. 1, pp. 121–124, IEEE, 2003.

[11] Baird, L., "Residual algorithms: Reinforcement learning with function approximation," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pp. 30–37, Citeseer, 1995.

[12] Barto, A., Sutton, R., and Anderson, C., "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on systems, man, and cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.

[13] Barto, A. G., Bradtke, S. J., and Singh, S. P., "Learning to act using real-time dynamic programming," *Artif. Intell.*, vol. 72, no. 1-2, pp. 81–138, 1995.

[14] Baxter, J., Tridgell, A., and Weaver, L., "Learning to play chess using temporal differences," *Machine Learning*, vol. 40, no. 3, pp. 243–263, 2000.

[15] Bellman, R., "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, 1957.

[16] Bhatnagar, S., Sutton, R., Ghavamzadeh, M., and Lee, M., "Incremental natural actor-critic algorithms," *Advances in neural information processing systems*, vol. 20, pp. 105–112, 2008.

[17] Blythe, J., "Decision-theoretic planning," *AI Magazine*, vol. 20, no. 2, p. 37, 1999.

[18] Boyan, J., *Learning Evaluation Functions for Global Optimization*. PhD thesis, Pittsburgh, PA, USA, 1998.

[19] Boyan, J. A. and Moore, A. W., "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems 7*, pp. 369–376, MIT Press, 1995.

[20] Busoniu, L., Ernst, D., De Schutter, B., and Babuska, R., "Policy search with cross-entropy optimization of basis functions," in *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pp. 153–160, IEEE, 2009.

[21] Busoniu, L., Ernst, D., De Schutter, B., and Babuska, R., "Cross-entropy optimization of control policies with adaptive basis functions," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 41, no. 1, pp. 196–209, 2011.

[22] Cheng, L. and Duran, M., "Logistics for world-wide crude oil transportation using discrete event simulation and optimal control," *Computers & chemical engineering*, vol. 28, no. 6-7, pp. 897–911, 2004.

[23] Chernova, S. and Veloso, M., "Interactive policy learning through confidence-based autonomy," *Journal of Artificial Intelligence Research*, vol. 34, no. 1, pp. 1–25, 2009.

[24] Choi, S. and Liu, J., "Optimal time-constrained trading strategies for autonomous agents," *Proc. of MAMA*, vol. 2000, 2000.

[25] Cichosz, P., "An analysis of experience replay in temporal difference learning," *Cybernetics and Systems*, vol. 30, no. 5, pp. 341–363, 1999.

[26] Clouse, J., "On integrating apprentice learning and reinforcement learning," *Electronic Doctoral Dissertations for UMass Amherst*, 1996.

[27] Cobo, L., Zang, P., Isbell, C., and Thomaz, A., "Automatic state abstraction from demonstration," in *In Proceedings of IJCAI-2011*, IJCAI-2011, 2011.

[28] Crites, R. and Barto, A., "Improving elevator performance using reinforcement learning," *Advances in Neural Information Processing Systems 8*, 1996.

[29] Dahl, F., "A Reinforcement Learning Algorithm Applied to Simplified Two-Player Texas Hold'em Poker," *Lecture Notes in Computer Science*, pp. 85–96, 2001.

[30] Dai, P. and Hansen, E., "Prioritizing Bellman backups without a priority queue," in *ICAPS*, 2007.

[31] De Farias, D. and Van Roy, B., "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003.

[32] de Farias, D. and Van Roy, B., "On constraint sampling in the linear programming approach to approximate dynamic programming," *Mathematics of Operations Research*, pp. 462–478, 2004.

[33] Dean, T. and Kanazawa, K., "A model for reasoning about persistence and causation," *Computational intelligence*, vol. 5, no. 2, pp. 142–150, 1989.

[34] Dietterich, T., "The MAXQ method for hierarchical reinforcement learning," in *Proceedings of the fifteenth international conference on machine learning*, pp. 118–126, Citeseer, 1998.

[35] Dimitrakakis, C. and Lagoudakis, M., "Rollout sampling approximate policy iteration," *Machine Learning*, vol. 72, no. 3, pp. 157–171, 2008.

[36] Dorigo, M. and Colombetti, M., "Robot shaping: developing autonomous agents through learning," *Artif. Intell.*, vol. 71, no. 2, pp. 321–370, 1994.

[37] Dvijotham, K. and Todorov, E., "A unifying framework for linearly solvable control," in *Uncertainty in Artificial Intelligence*, 2011.

[38] Dzeroski, S., De Raedt, L., and Driessens, K., "Relational reinforcement learning," *Machine Learning, 43*, vol. 1, no. 2, pp. 7–52, 2001.

[39] Falk, R. and Konold, C., "Making sense of randomness: Implicit encoding as a basis for judgment.," *Psychological Review*, vol. 104, no. 2, p. 301, 1997.

[40] Farahmand, A., Ghavamzadeh, M., Szepesvari, C., and Mannor, S., "Regularized policy iteration," *Advances in Neural Information Processing Systems*, vol. 21, pp. 441–448, 2009.

[41] Fern, A., Yoon, S., and Givan, R., "Approximate policy iteration with a policy language bias," *Advances in neural information processing systems*, vol. 16, no. 3, pp. 847–854, 2004.

[42] Fern, A., Yoon, S., and Givan, R., "Learning domain-specific control knowledge from random walks," in *International Conference on Automated Planning and Scheduling*, pp. 191–199, 2004.

[43] Finnsson, H. and Bj
"ornsson, Y., "Simulation-based approach to general game playing," in *Proceedings of the 23rd national conference on Artificial intelligence*, pp. 259–264, 2008.

[44] Galway, L., Charles, D., and Black, M., "Machine learning in digital games: a survey," *Artificial Intelligence Review*, vol. 29, no. 2, pp. 123–161, 2008.

[45] Geist, M. and Pietquin, O., "A brief survey of parametric value function approximation," tech. rep., Technical report, Supélec, 2010.

[46] Gelly, S. and Wang, Y., "Exploration exploitation in go: Uct for monte-carlo go," in *Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006)*, Citeseer, 2006.

[47] Gomez, F., Schmidhuber, J., and Miikkulainen, R., "Accelerated neural evolution through cooperatively coevolved synapses," *The Journal of Machine Learning Research*, vol. 9, pp. 937–965, 2008.

[48] Gomez, F. J. and Miikkulainen, R., "Active guidance for a finless rocket using neuroevolution," in *Genetic Evolutionary Computation Conference*, 2003.

[49] Gordon, G., "Stable Function Approximation in Dynamic Programming," in *Proceedings of the Twelfth International Conference on Machine Learning*, p. 261, Morgan Kaufmann, 1995.

[50] Gordon, G., *Approximate solutions to Markov decision processes*. PhD thesis, Citeseer, 1999.

[51] Grollman, D. and Jenkins, O., "Dogged learning for robots," in *IEEE International Conference on Robotics and Automation*, pp. 2483–2488, Citeseer, 2007.

[52] Guestrin, C., Koller, D., Parr, R., and Venkataraman, S., "Efficient solution algorithms for factored MDPs," *Journal of Artificial Intelligence Research*, vol. 19, no. 10, pp. 399–468, 2003.

[53] Hall, M., *Correlation-based feature selection for machine learning*. PhD thesis, 1999.

[54] Heidrich-Meisner, V. and Igel, C., "Hoeffding and bernstein races for selecting policies in evolutionary direct policy search," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ACM New York, NY, USA, 2009.

[55] Heidrich-Meisner, V. and Igel, C., "Uncertainty handling cma-es for reinforcement learning," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 1211–1218, ACM, 2009.

[56] Hengst, B., "Discovering hierarchy in reinforcement learning with hexq," 2002.

[57] Hoey, J., St-Aubin, R., Hu, A., and Boutilier, C., "Spudd: Stochastic planning using decision diagrams," in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 279–288, Citeseer, 1999.

[58] Howard, R., "Dynamic programming and markov decision processes," 1960.

[59] Hu, J., Fu, M., Ramezani, V., and Marcus, S., "An evolutionary random policy search algorithm for solving markov decision processes," *INFORMS Journal on Computing*, vol. 19, no. 2, p. 161, 2008.

[60] Irani, A., Zang, P., and Isbell, C. L., "Scaling reinforcement learning through fast propagation of optimal value information," in *Submitted to Proceedings of Autonomous Agents and Multi-Agent Systems*, 2010.

[61] Isbell, C., Shelton, C., Kearns, M., Singh, S., and Stone, P., "A social reinforcement learning agent," in *International Conference on Autonomous agents*, pp. 377–384, ACM New York, NY, USA, 2001.

[62] Jonsson, A. and Barto, A., "Causal graph based decomposition of factored mdps," *J. Mach. Learn. Res.*, vol. 7, pp. 2259–2301, 2006.

[63] Kakade, S., "A natural policy gradient," in *Proceedings of NIPS*, vol. 14, 2001.

[64] Kearns, M., Mansour, Y., and Ng, A., "A sparse sampling algorithm for near-optimal planning in large Markov decision processes," *Machine Learning*, vol. 49, no. 2, pp. 193–208, 2002.

[65] Keller, P., Mannor, S., and Precup, D., "Automatic basis function construction for approximate dynamic programming and reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, pp. 449–456, ACM, 2006.

[66] Knox, W. and Stone, P., "Interactively shaping agents via human reinforcement: The tamer framework," in *Proceedings of the fifth international conference on Knowledge capture*, pp. 9–16, ACM, 2009.

[67] Knox, W. and Stone, P., "Combining manual feedback with subsequent mdp reward signals for reinforcement learning," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 5–12, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[68] Kocsis, L. and Szepesvári, C., "Bandit based monte-carlo planning," *Machine Learning: ECML 2006*, pp. 282–293, 2006.

[69] Kohl, N. and Stone, P., "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3, pp. 2619–2624, IEEE, 2004.

[70] Koller, D. and Parr, R., "Policy iteration for factored mdps," in *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00*, Citeseer, 2000.

[71] Kolter, J. and Ng, A., "Regularization and feature selection in least-squares temporal difference learning," in *International Conference on Machine Learning*, pp. 521–528, ACM, 2009.

[72] Konda, V. and Tsitsiklis, J., "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2004.

[73] Kuniyoshi, Y., Inaba, M., and Inoue, H., "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 6, pp. 799–822, 1994.

[74] Lagoudakis, M. G. and Parr, R., "Least-squares policy iteration," pp. 1107–1149, 2003.

[75] Lazaric, A., Ghavamzadeh, M., and Munos, R., "Analysis of a classification-based policy iteration algorithm," in *Proceedings 27th International Conference on Machine Learning (ICML-10), Haifa, Israel*, pp. 21–24, Citeseer, 2010.

[76] Li, X., Shen, X., Jing, Y., and Zhang, S., "Simulated annealing-reinforcement learning algorithm for abr traffic control of atm networks," in *Decision and Control, 2007 46th IEEE Conference on*, pp. 5716–5721, IEEE, 2007.

[77] Littman, M., "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the eleventh international conference on machine learning*, vol. 157, p. 163, Citeseer, 1994.

[78] Littman, M., Dean, T., and Kaelbling, L., "On the complexity of solving Markov decision problems," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 394–402, Citeseer, 1995.

[79] Loh, W., "Regression trees with unbiased variable selection and interaction detection," *Statistica Sinica*, vol. 12, pp. 361–386, 2002.

[80] Maei, H. and Sutton, R., "Gq (λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces," in *AGI*, pp. 91–96, Citeseer, 2010.

[81] Maei, H., Szepesvári, C., Bhatnagar, S., Precup, D., Silver, D., and Sutton, R., "Convergent temporal-difference learning with arbitrary smooth function approximation," *Advances in Neural Information Processing Systems*, vol. 22, pp. 1204–1212, 2009.

[82] Mahadevan, S., "Representation policy iteration," in *Proceedings of the 21st International Conference on Uncertainty in Artificial Intelligence*, Citeseer, 2005.

[83] Mannor, S., Rubinstein, R., and Gat, Y., "The cross entropy method for fast policy search," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, vol. 20, p. 512, 2003.

[84] Mannor, S., Menache, I., Hoze, A., and Klein, U., "Dynamic abstraction in reinforcement learning via clustering," in *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, (New York, NY, USA), p. 71, ACM, 2004.

[85] McGovern, A. and Barto, A. G., "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proc. 18th International Conf. on Machine Learning*, pp. 361–368, Morgan Kaufmann, San Francisco, CA, 2001.

[86] Minkoff, A., "A markov decision model and decomposition heuristic for dynamic vehicle dispatching," *Operations Research*, pp. 77–90, 1993.

[87] Munos, R. and Szepesvari, C., "Finite time bounds for sampling based fitted value iteration," *Journal of Machine Learning Research*, 2005.

[88] N. Mehta, M. Wynkoop, S. R., Tadepalli, P., and Dietterich, T., "Automatic induction of maxq hierarchies," in *Proceedings of the Hierarchical Organization of Behavior Workshop*, 21st Conference on Neural Information Processing Systems, 2007.

[89] Nakanishi, J., Morimoto, J., Endo, G., Cheng, G., Schaal, S., and Kawato, M., "Learning from demonstration and adaptation of biped locomotion," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 79–91, 2004.

[90] Nevmyvaka, Y., Feng, Y., and Kearns, M., "Reinforcement learning for optimized trade execution," in *Proceedings of the 23rd international conference on Machine learning*, p. 680, ACM, 2006.

[91] Ng, A. Y., Harada, D., and Russell, S. J., "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 16, pp. 278–287, Morgan Kaufmann, 1999.

[92] Ng, A. and Jordan, M., "Pegasus: A policy search method for large mdps and pomdps," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pp. 406–415, Citeseer, 2000.

[93] Ng, A., Kim, H., Jordan, M., Sastry, S., and Ballianda, S., "Autonomous helicopter flight via reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 16, 2004.

[94] Ng, A. and Russell, S., "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 663–670, 2000.

[95] Papadimitriou, C. and Tsitsiklis, J., "The complexity of Markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[96] Parkes, D. and Singh, S., "An mdp-based approach to online mechanism design," in *Proc. 17th Annual Conf. on Neural Information Processing Systems (NIPS03)*, Citeseer, 2003.

[97] Parr, R., Painter-Wakefield, C., Li, L., and Littman, M., "Analyzing feature generation for value-function approximation," in *International Conference on Machine learning*, pp. 737–744, ACM, 2007.

[98] Parr, R. and Russell, S., "Reinforcement learning with hierarchies of machines," in *Advances in Neural Information Processing Systems* (Jordan, M. I., Kearns, M. J., and Solla, S. A., eds.), vol. 10, The MIT Press, 1997.

[99] P.D., B. and J.N., T., *Neuro-Dynamic Programming*. Nashua, NH: Athena Scientific, 1996.

[100] Peters, J. and Schaal, S., "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, China*, pp. 2219–2225, Citeseer, 2006.

[101] Peters, J., Vijayakumar, S., and Schaal, S., "Natural actor-critic," *Machine Learning: ECML 2005*, pp. 280–291, 2005.

[102] Pickett, M. and Barto, A. G., "Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning," in *ICML 2002: Proceedings of the Nineteenth International Conference on Machine Learning*, (San Francisco, CA, USA), pp. 506–513, Morgan Kaufmann Publishers Inc., 2002.

[103] Polanyi, M. and MyiLibrary, *Personal knowledge: Towards a post-critical philosophy*. Routledge & Kegan Paul London, 1962.

[104] Price, B. and Boutilier, C., "Accelerating reinforcement learning through implicit imitation," *Journal of Artificial Intelligence Research*, vol. 19, pp. 569–629, 2003.

[105] Puterman, M., *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, Inc., 1994.

[106] Randløv, J. and Alstrøm, P., "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 463–471, Citeseer, 1998.

[107] Rummery, G. and Niranjan, M., *On-line Q-learning using connectionist systems*. Citeseer, 1994.

[108] Sammut, C., Hurst, S., Kedzier, D., and Michie, D., "Learning to fly," in *Proceedings of the ninth international workshop on Machine learning*, pp. 385–393, Citeseer, 1992.

[109] Samuel, A. L., "Some studies in machine learning using the game of checkers," *IBM Journal on Research and Development*, pp. 210–229, 1959.

[110] Schaal, S. and Atkeson, C., "Robot juggling: An implementation of memory-based learning," *Control Systems Magazine*, vol. 14, no. 1, pp. 57–71, 1994.

[111] Schaal, S., "Learning from demonstration," in *Advances in Neural Information Processing Systems 9*, MIT Press, 1997.

[112] Schweitzer, P. and Seidmann, A., "Generalized polynomial approximations in markovian decision processes," *Journal of mathematical analysis and applications*, vol. 110, no. 2, pp. 568–582, 1985.

[113] Scott, D., "Parametric statistical modeling by minimum integrated square error," *Technometrics*, vol. 43, no. 3, pp. 274–285, 2001.

[114] Silver, D., Sutton, R., and M
"uller, M., "Reinforcement learning of local shape in the game of Go," in *20th International Joint Conference on Artificial Intelligence*, pp. 1053–1058, 2007.

[115] Singh, S., Jaakkola, T., and Jordan, M., "Learning without state-estimation in partially observable Markovian decision processes," in *International Conference on Machine Learning*, pp. 284–292, 1994.

[116] Smart, W. and Kaelbling, L., "Effective reinforcement learning for mobile robots," in *Proceedings- IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3404–3410, Citeseer, 2002.

[117] Song, H., Liu, C., Lawarree, J., and Dahlgren, R., "Optimal electricity supply bidding by markov decision process," *Power Systems, IEEE Transactions on*, vol. 15, no. 2, pp. 618–624, 2000.

[118] Stone, P. and Sutton, R., "Scaling reinforcement learning toward RoboCup soccer," in *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pp. 537–544, Citeseer, 2001.

[119] Sturtevant, N. and White, A., "Feature construction for reinforcement learning in hearts," *Lecture Notes in Computer Science*, vol. 4630, p. 122, 2007.

[120] Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[121] Sutton, R. S., Precup, D., and Singh, S. P., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[122] Sutton, R. S., Sutton, R. S., Precup, D., and Singh, S. P., "Intra-option learning about temporally abstract actions," in *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, (San Francisco, CA, USA), pp. 556–564, Morgan Kaufmann Publishers Inc., 1998.

[123] Sutton, R., "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, pp. 1038–1044, 1996.

[124] Sutton, R., Maei, H., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E., "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 993–1000, ACM, 2009.

[125] Sutton, R., McAllester, D., Singh, S., and Mansour, Y., "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, no. 22, 2000.

[126] Szita, I. and L
"orincz, A., "Learning tetris using the noisy cross-entropy method," *Neural Computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[127] Taylor, M. and Stone, P., "Transfer learning for reinforcement learning domains: A survey," *The Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[128] Tesauro, G., "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, vol. 6, no. 2, pp. 215–219, 1994.

[129] Thomaz, A. and Breazeal, C., "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, p. 1000, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[130] Thomaz, A. and Breazeal, C., "Transparency and socially guided machine learning," in *5th Intl. Conf. on Development and Learning (ICDL)*, 2006.

[131] Thomaz, A. and Breazeal, C., "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 716–737, 2008.

[132] Thorpe, T., "Vehicle traffic light control using sarsa," in *Online]. Available: citeseer. ist. psu. edu/thorpe97vehicle. html*, Citeseer, 1997.

[133] Toussaint, M., Harmeling, S., and Storkey, A., "Probabilistic inference for solving (po) mdps," *Institute for Adaptive and Neural Computation*, 2006.

[134] Trick, M. and S., Z., "Spline approximations to value functions: A linear programming approach," *Macroeconomic Dynamics*, vol. 1, pp. 255–277, 1997.

[135] Tsitsiklis, J. and Roy, B., "Feature-based methods for large scale dynamic programming," *Machine Learning*, vol. 22, no. 1, pp. 59–94, 1996.

[136] Tsitsiklis, J. and Van Roy, B., "An analysis of temporal-difference learning with function approximation," *Automatic Control, IEEE Transactions on*, vol. 42, no. 5, pp. 674–690, 1997.

[137] Tsitsiklis, J. and Van Roy, B., "Regression methods for pricing complex american-style options," *Neural Networks, IEEE Transactions on*, vol. 12, no. 4, pp. 694–703, 2001.

[138] Wagenaar, W., "Generation of random sequences by human subjects: A critical survey of literature.," *Psychological Bulletin*, vol. 77, no. 1, p. 65, 1972.

[139] Walsh, T., Goschin, S., and Littman, M., "Integrating sample-based planning and model-based reinforcement learning," in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, 2010.

[140] Watkins, C. and Dayan, P., "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[141] Williams, R., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[142] Wingate, D., "Solving large mdp quickly with partitioned value iteration," Master's thesis, 2004.

[143] Wingate, D., Goodman, N. D., Roy, D. M., Kaelbling, L. P., and Tenenbaum, J. B., "Bayesian policy search with policy priors," in *International Joint Conference on Artificial Intelligence*, 2011.

[144] Wingate, D. and Seppi, K. D., "Prioritization methods for accelerating mdp solvers," *Journal of Machine Learning Research*, vol. 6, pp. 851–881, 2005.

[145] Witten, I., "An adaptive optimal controller for discrete-time markov environments," *Information and Control*, vol. 34, no. 4, pp. 286–295, 1977.

[146] Xu, X., Hu, D., and Lu, X., "Kernel-based least squares policy iteration for reinforcement learning," *Neural Networks, IEEE Transactions on*, vol. 18, no. 4, pp. 973–992, 2007.

[147] Yoon, S., Fern, A., and Givan, R., "Learning heuristic functions from relaxed plans," in *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 162–171, 2006.

[148] Zang, P., Irani, A., Zhou, P., Isbell Jr, C., and Thomaz, A., "Using training regimens to teach expanding function approximators," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 341–348, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[149] Zang, P., Zhou, P., Minnen, D., and Isbell, C., "Discovering options from example trajectories," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ACM New York, NY, USA, 2009.

[150] Zang, P., Irani, A., and Isbell, C. L., "Horizon-based value iteration," tech. rep., 2007.

[151] Zhang, W. and Dietterich, T., "High-performance job-shop scheduling with a timedelay td () network," *Advances in neural information processing systems*, vol. 8, pp. 1024–1030, 1996.